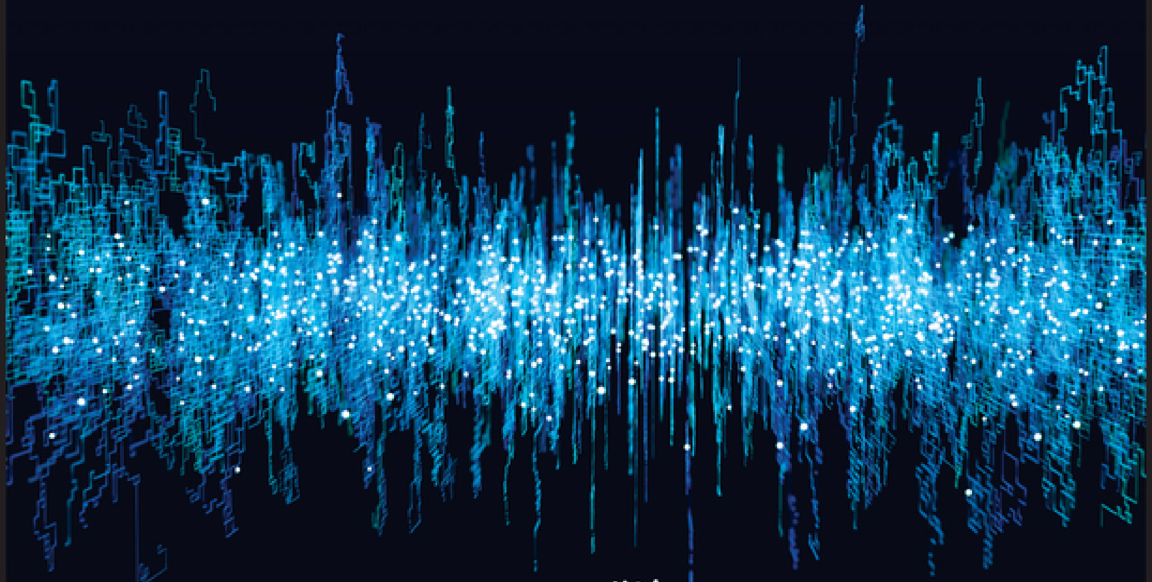
Second Edition

An Introduction to Audio Content Analysis

Music Information Retrieval Tasks & Applications

Alexander Lerch



with website

IEEE PRESS

WILEY

An Introduction to Audio Content Analysis

IEEE Press
445 Hoes Lane
Piscataway, NJ 08854

IEEE Press Editorial Board
Sarah Spurgeon, *Editor in Chief*

Jón Atli Benediktsson
Anjan Bose
Adam Drobot
Peter (Yong) Lian
Andreas Molisch
Saeid Nahavandi
Jeffrey Reed
Thomas Robertazzi
Diomidis Spinellis
Ahmet Murat Tekalp

An Introduction to Audio Content Analysis

Music Information Retrieval Tasks & Applications

Second Edition

Alexander Lerch
Georgia Institute of Technology
Atlanta, USA

IEEE PRESS
WILEY

Copyright © 2023 by The Institute of Electrical and Electronics Engineers, Inc. All rights reserved.

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

Edition History
Wiley (1e, 2012)

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4470, or on the web at www.copyright.com. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at http://www.wiley.com/go/permission.

Trademarks: Wiley and the Wiley logo are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates in the United States and other countries and may not be used without written permission. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc. is not associated with any product or vendor mentioned in this book.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Neither the publisher nor author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services or for technical support, please contact our Customer Care Department within the United States at (800) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic formats. For more information about Wiley products, visit our web site at www.wiley.com.

Library of Congress Cataloging-in-Publication Data

Names: Lerch, Alexander, author. | John Wiley & Sons, publisher.
Title: An introduction to audio content analysis : music information retrieval tasks & applications / Alexander Lerch.
Description: Second edition. | Hoboken, New Jersey : Wiley-IEEE Press, [2023] | Includes bibliographical references and index.
Identifiers: LCCN 2022046787 (print) | LCCN 2022046788 (ebook) | ISBN 9781119890942 (cloth) | ISBN 9781119890966 (adobe pdf) | ISBN 9781119890973 (epub)
Subjects: LCSH: Computer sound processing. | Computational auditory scene analysis. | Content analysis (Communication)–Data processing.
Classification: LCC TK7881.4 .L486 2023 (print) | LCC TK7881.4 (ebook) | DDC 006.4/5–dc23/eng/20221018
LC record available at https://lccn.loc.gov/2022046787
LC ebook record available at https://lccn.loc.gov/2022046788

Cover image: © Weiquan Lin/Getty Images
Cover design by Wiley

Set in 9.5/12.5pt STIXTwoText by Straive, Chennai, India

For Mila

Contents

Author Biography *xvii*
Preface *xix*
Acronyms *xxi*
List of Symbols *xxv*
Source Code Repositories *xxix*

1 Introduction *1*
1.1 A Short History of Audio Content Analysis *3*
1.2 Applications and Use Cases *4*
1.2.1 Music Browsing and Music Discovery *4*
1.2.2 Music Consumption *4*
1.2.3 Music Production *5*
1.2.4 Music Education *5*
1.2.5 Generative Music *6*
References *6*

Part I Fundamentals of Audio Content Analysis *9*

2 Analysis of Audio Signals *11*
2.1 Audio Content *11*
2.2 Audio Content Analysis Process *13*
2.3 Exercises *15*
2.3.1 Questions *15*
References *15*

3 Input Representation *17*
3.1 Audio Signals *17*
3.1.1 Periodic Signals *18*
3.1.2 Random Signals *20*
3.1.3 Statistical Signal Description *20*

3.1.3.1 Arithmetic Mean *22*
3.1.3.2 Geometric Mean *22*
3.1.3.3 Harmonic Mean *23*
3.1.3.4 Variance and Standard Deviation *23*
3.1.3.5 Quantiles and Quantile Ranges *24*
3.1.4 Digital Audio Signals *25*
3.2 Audio Preprocessing *26*
3.2.1 Down-Mixing *26*
3.2.2 DC Removal *27*
3.2.3 Normalization *28*
3.2.4 Sample Rate Conversion *29*
3.2.5 Block-Based Processing *29*
3.2.6 Other Preprocessing Options *32*
3.3 Time-Frequency Representations *32*
3.3.1 Fourier Transform *32*
3.3.2 Constant Q Transform *35*
3.3.3 Log-Mel Spectrogram *36*
3.3.4 Filterbanks *37*
3.4 Other Input Representations *39*
3.5 Instantaneous Features *39*
3.5.1 Spectral Centroid *43*
3.5.2 Spectral Spread *45*
3.5.3 Spectral Skewness and Spectral Kurtosis *46*
3.5.4 Spectral Rolloff *48*
3.5.5 Spectral Decrease *50*
3.5.6 Spectral Slope *50*
3.5.7 Mel Frequency Cepstral Coefficients *51*
3.5.8 Spectral Flux *55*
3.5.9 Spectral Crest Factor *57*
3.5.10 Spectral Flatness *58*
3.5.11 Tonal Power Ratio *60*
3.5.12 Maximum of Autocorrelation Function *62*
3.5.13 Zero Crossing Rate *64*
3.6 Learned Features *65*
3.7 Feature PostProcessing *67*
3.7.1 Derived Features *67*
3.7.2 Feature Aggregation *68*
3.7.3 Normalization and Mapping *69*
3.7.4 Feature Dimensionality Reduction *72*
3.7.4.1 Feature Subset Selection *73*
3.7.4.2 Feature Space Transformation *75*
3.8 Exercises *76*

3.8.1 Questions *76*
3.8.2 Assignments *78*
References *81*

4 Inference *91*
4.1 Classification *92*
4.2 Regression *97*
4.3 Clustering *99*
4.4 Distance and Similarity *101*
4.5 Underfitting and Overfitting *102*
4.6 Exercises *103*
4.6.1 Questions *103*
4.6.2 Assignments *103*
References *104*

5 Data *107*
5.1 Data Split *109*
5.1.1 N-Fold Cross Validation *110*
5.2 Training Data Augmentation *111*
5.3 Utilization of Data From Related Tasks *113*
5.4 Reducing Accuracy Requirements for Data Annotation *114*
5.5 Semi-, Self-, and Unsupervised Learning *115*
5.6 Exercises *116*
5.6.1 Questions *116*
5.6.2 Assignments *116*
References *116*

6 Evaluation *119*
6.1 Metrics *121*
6.1.1 Classification *121*
6.1.2 Regression *124*
6.1.3 Clustering *125*
6.2 Exercises *126*
6.2.1 Questions *126*
References *126*

Part II Music Transcription *127*

7 Tonal Analysis *129*
7.1 Human Perception of Pitch *129*
7.1.1 Pitch Scales *129*
7.1.2 Chroma Perception *132*
7.2 Representation of Pitch in Music *133*

7.2.1 Pitch Classes and Names 133
7.2.2 Intervals 134
7.2.3 The Frequency of Musical Pitch 135
7.2.3.1 Temperament 136
7.2.3.2 Intonation 137
7.3 Fundamental Frequency Detection 138
7.3.1 Detection Accuracy 139
7.3.1.1 Time Domain 139
7.3.1.2 Frequency Domain 140
7.3.1.3 Potential Solutions 141
7.3.2 Preprocessing 143
7.3.3 Monophonic Input Signals 143
7.3.3.1 Zero Crossing Rate 144
7.3.3.2 Autocorrelation Function 145
7.3.3.3 Average Magnitude Difference Function 147
7.3.3.4 Harmonic Product Spectrum and Harmonic Sum Spectrum 149
7.3.3.5 Autocorrelation Function of the Magnitude Spectrum 152
7.3.3.6 Cepstral Pitch Detection 152
7.3.3.7 Maximum Likelihood and Template Matching 154
7.3.3.8 Auditory-Motivated Pitch Tracking 155
7.3.4 Polyphonic Input Signals 156
7.3.4.1 Iterative Subtraction 157
7.3.4.2 Nonnegative Matrix Factorization 159
7.3.4.3 Other Approaches 163
7.3.5 Evaluation 164
7.3.5.1 Metrics 164
7.3.5.2 Datasets 166
7.3.5.3 Results 166
7.4 Tuning Frequency Estimation 166
7.4.1 Approaches to Tuning Frequency Estimation 168
7.4.2 Evaluation 170
7.5 Key Detection 170
7.5.1 Pitch Chroma 173
7.5.1.1 Pitch Chroma Properties 178
7.5.1.2 Features Derived from the Pitch Chroma 179
7.5.2 Approaches to Key Detection 180
7.5.2.1 Key Profiles 181
7.5.2.2 Similarity Measure between Template and Extracted Vector 182
7.5.3 Evaluation 183
7.5.3.1 Metrics 184
7.5.3.2 Datasets 184
7.5.3.3 Results 184

7.6 Chord Recognition 185
7.6.1 Approaches to Chord Recognition 185
7.6.2 Viterbi Algorithm 188
7.6.3 Evaluation 192
7.6.3.1 Metrics 193
7.6.3.2 Datasets 193
7.6.3.3 Results 193
7.7 Exercises 194
7.7.1 Questions 194
7.7.2 Assignments 196
References 201

8 Intensity 217
8.1 Human Perception of Intensity and Loudness 217
8.2 Representation of Dynamics in Music 219
8.3 Features 220
8.3.1 Root Mean Square 220
8.3.2 Weighted Root Mean Square 221
8.3.3 Peak Envelope 223
8.3.4 Psycho-Acoustic Loudness Features 225
8.4 Exercises 225
8.4.1 Questions 225
8.4.2 Assignments 226
References 227

9 Temporal Analysis 229
9.1 Human Perception of Temporal Events 229
9.1.1 Onsets 229
9.1.2 Tempo and Meter 232
9.1.3 Rhythm 233
9.1.4 Timing 234
9.2 Representation of Temporal Events in Music 234
9.2.1 Tempo and Time Signature 235
9.2.2 Note Value 235
9.3 Onset Detection 236
9.3.1 Novelty Function 236
9.3.2 Peak Picking 239
9.3.3 Evaluation 241
9.3.3.1 Metrics 241
9.3.3.2 Datasets 243
9.3.3.3 Results 243
9.4 Beat Histogram 243
9.4.1 Beat Histogram Features 245

9.5 Detection of Tempo and Beat Phase *245*
9.5.1 Evaluation *249*
9.5.1.1 Metrics *249*
9.5.1.2 Datasets *250*
9.5.1.3 Results *250*
9.6 Detection of Meter and Downbeat *250*
9.7 Structure Detection *252*
9.7.1 Self-Similarity Matrix *253*
9.7.2 Approaches to Structure Detection *256*
9.7.2.1 Novelty Analysis *256*
9.7.2.2 Homogeneity Analysis *256*
9.7.2.3 Repetition Analysis *256*
9.7.3 Evaluation *258*
9.7.3.1 Metrics *259*
9.7.3.2 Datasets *259*
9.7.3.3 Results *260*
9.8 Automatic Drum Transcription *260*
9.8.1 Transcription of Drum Onsets *261*
9.8.2 Evaluation *262*
9.9 Exercises *262*
9.9.1 Questions *262*
9.9.2 Assignments *263*
References *266*

10 Alignment *281*
10.1 Dynamic Time Warping *281*
10.1.1 Example *286*
10.1.2 Common Variants *287*
10.1.3 Optimizations *288*
10.2 Audio-to-Audio Alignment *289*
10.3 Audio-to-Score Alignment *291*
10.3.1 Real-Time Systems *292*
10.3.2 Non-Real-Time Systems *293*
10.4 Evaluation *294*
10.4.1 Metrics *294*
10.4.2 Data *295*
10.5 Exercises *296*
10.5.1 Questions *296*
10.5.2 Assignments *296*
References *298*

Part III Music Identification, Classification, and Assessment *303*

11 **Audio Fingerprinting** *305*
11.1 Fingerprint Extraction *307*
11.2 Fingerprint Matching *308*
11.3 Fingerprinting System: Example *309*
11.4 Evaluation *312*
References *312*

12 **Music Similarity Detection and Music Genre Classification** *317*
12.1 Music Similarity Detection *317*
12.1.1 Approaches to Music Similarity Computation *318*
12.1.2 Evaluation *319*
12.2 Musical Genre Classification *319*
12.2.1 Approaches to Musical Genre Classification *321*
12.2.2 Genre Classification: Example *324*
12.2.3 Evaluation *325*
12.2.3.1 Metrics *326*
12.2.3.2 Data *326*
12.2.3.3 Results *326*
12.2.4 Exercises *326*
12.2.5 Questions *326*
12.2.6 Assignments *327*
References *328*

13 **Mood Recognition** *337*
13.1 Approaches to Mood Recognition *338*
13.2 Evaluation *341*
References *342*

14 **Musical Instrument Recognition** *347*
14.1 Evaluation *349*
References *350*

15 **Music Performance Assessment** *355*
15.1 Music Performance *355*
15.2 Music Performance Analysis *357*
15.3 Approaches to Music Performance Assessment *358*
References *360*

Part IV Appendices *365*

Appendix A Fundamentals *367*
A.1 Sampling and Quantization *367*
A.1.1 Sampling *367*
A.1.2 Quantization *369*
A.2 Convolution *372*
A.2.1 Identity *372*
A.2.2 Commutativity *373*
A.2.3 Associativity *373*
A.2.4 Distributivity *374*
A.2.5 Circularity *374*
A.2.6 Simple Filter Examples *375*
A.2.6.1 Moving Average Filter *375*
A.2.6.2 Single-Pole Low-Pass Filter *376*
A.2.7 Zero-Phase Filtering with IIRs *377*
A.3 Correlation Function *378*
A.3.1 Normalization *379*
A.3.2 Autocorrelation Function *380*
A.3.3 Applications *380*
A.3.4 Calculation in the Frequency Domain *381*
A.3.4.1 Frequency Domain Compression *382*
References *382*

Appendix B Fourier Transform *385*
B.1 Properties of the Fourier Transformation *386*
B.1.1 Inverse Fourier Transform *386*
B.1.2 Superposition *386*
B.1.3 Convolution and Multiplication *386*
B.1.4 Parseval's Theorem *387*
B.1.5 Time and Frequency Shift *388*
B.1.6 Symmetry *388*
B.1.7 Time and Frequency Scaling *390*
B.1.8 Derivatives *390*
B.2 Spectrum of Example Time Domain Signals *390*
B.2.1 Delta Function *390*
B.2.2 Constant *391*
B.2.3 Cosine *391*
B.2.4 Rectangular Window *391*
B.2.5 Delta Pulse *392*
B.3 Transformation of Sampled Time Signals *392*

B.4 Short Time Fourier Transform of Continuous Signals *393*
B.4.1 Window Functions *395*
B.4.1.1 Rectangular Window *395*
B.4.1.2 Bartlett Window *396*
B.4.1.3 Generalized Superposed Cosines *396*
B.4.1.4 Generalized Power of Cosine *396*
B.5 Discrete Fourier Transform *397*
B.5.1 Window Functions *398*
B.5.1.1 Discrete Window Properties *398*
B.5.2 Fast Fourier Transform *399*
B.6 Frequency Reassignment: Instantaneous Frequency *399*
References *402*

Appendix C Principal Component Analysis *405*
C.1 Computation of the Transformation Matrix *406*
C.2 Interpretation of the Transformation Matrix *407*

Appendix D Linear Regression *409*

Appendix E Software for Audio Analysis *411*
E.1 Frameworks and Libraries *412*
E.1.1 librosa *412*
E.1.2 Essentia *412*
E.1.3 openSMILE *412*
E.1.4 Marsyas *413*
E.1.5 jMIR *413*
E.1.6 MIRtoolbox *413*
E.1.7 Yaafe *413*
E.1.8 madmom *413*
E.1.9 Software for Education *414*
E.1.10 Other Software *414*
E.2 Data Annotation and Visualization *414*
References *415*

Appendix F Datasets *417*
References *419*

Index *425*

Author Biography

Alexander Lerch is Associate Professor at the School of Music, Georgia Institute of Technology. He studied Electrical Engineering at the Berlin Institute of Technology and Tonmeister (music production) at the University of Arts, Berlin, and received his PhD (Audio Communications) from the Berlin Institute of Technology in the year 2008. He currently leads the Georgia Tech Music Informatics Group with a research focus on the design and implementation of machine learning and signal processing methods targeting audio and music in the areas audio content analysis, music information retrieval, machine listening, and music meta-creation. Lerch has served as co-chair of ISMIR2021, the 22nd International Society for Music Information Retrieval Conference.

Before joining Georgia Tech, he co-founded the company zplane.development, an industry leader providing advanced music technology to the music industry. zplane technologies are nowadays used by millions of musicians and producers world-wide in a wide variety of products.

Preface

The fields of Audio Content Analysis (ACA) and Music Information Retrieval (MIR) have seen rapid growth over the past decade as indicated by a rising number in publications, growing conference attendance, and an increasing number of commercial applications in the field. This growth is driven by the need to intelligently browse, retrieve, and process the large amount of music data at our fingertips and is also reflected by growing interest of students, engineers, and software developers to learn about methods for the analysis of audio signals.

Inspired by the feedback received from readers of the first edition of this book, my goals for the second edition were threefold: (i) remain focused on the introduction of baseline systems rather than state-of-the-art approaches, (ii) widen the scope of tasks introduced, and (iii) provide additional materials enhancing the learning experience. While the baseline systems presented in this book cannot be considered state-of-the-art, their knowledge is, in my opinion, crucial for an introduction into and understanding of the field. Not only does this allow us to learn about the wide variety of tasks that mirror the variety of content we handle in the analysis of musical audio, but by looking at each of these tasks, we gain insights into the task-specific challenges and problems, thus gaining an understanding of inherent principles and challenges as well as parameters of interest. In addition, baseline systems allow for hands-on exercises and provide an important reference data point clarifying the expected performance of an analysis system.

The second edition of this book comes with noticeable changes from the first edition. Some parts have been shortened by removing unnecessary detail, while others have been extended to provide more useful detail. In addition, evaluation methodologies are summarized for most tasks; while evaluation often seems like a chore and uninteresting part of designing a system, it is a crucial part: without evaluation, no one knows how well a system works. Most chapters now also conclude with a set of questions and assignments assessing the learning outcomes. As this text intends to provide a starting point and guidance for the design of novel musical audio content analysis systems, every chapter contains many references to relevant work. Last but not least, several new topics are introduced, and the whole text has been restructured into three main parts plus an appendix.

The first part covers fundamentals shared by many audio analysis systems. It starts with a closer look at musical audio content and a general flowchart of an audio content analysis system. After a quick coverage of signals and general preprocessing concepts, typical time–frequency representations are introduced, which nowadays serve as the input representations of many analysis systems. A closer look at the most commonly used instantaneous or low-level features is then followed by an introduction of feature postprocessing methods covering normalization, aggregation, as well as general dimensionality reduction methods. The second part introduces systems that transcribe individual musical properties of the audio signal. These are mostly properties that are more or less explicitly defined in a musical score, such as pitch, dynamics, and tempo. More specifically, these approaches are grouped into the content dimensions *tonal* (fundamental frequency, tuning frequency, musical key, and chords), *intensity* (level and loudness), and *temporal analysis* (onsets, beats, tempo, meter, and structure). Audio analysis approaches related to classification and identification are presented in Part Three. Many of these systems do not focus on individual musical content dimensions but aim at extracting high-level content from the audio signal. Thus, the classifications of musical genre and mood, as well as the assessment of student music performances, are introduced. In addition, audio fingerprinting is discussed as one of the audio analysis technologies arguably most impactful for the consumer in the past. The appendix provides a more in-depth reference to basic concepts such as audio sampling and quantization, the Fourier transform, introduces correlation and convolution, as well as providing quick references to principal component analysis and linear regression.

Note that this edition also comes with a tighter integration with online resources such as slides and example code in both MATLAB and Python, as well as a website with resources such as a list of available datasets. The entry point for all these additional (and freely available) resources is https://www.audiocontentanalysis.org.

Atlanta, Georgia *Alexander Lerch*

Acronyms

ACA	Audio Content Analysis
ACF	Autocorrelation Function
ADT	Automatic Drum Transcription
AMDF	Average Magnitude Difference Function
ANN	Artificial Neural Network
AOT	Acoustic Onset Time
AUC	Area Under Curve
BPM	Beats per Minute
CCF	Cross Correlation Function
CCIR	Comité Consultatif International des Radiocommunications
CiCF	Circular Correlation Function
CD	Compact Disc
COG	Center of Gravity
CQT	Constant Q Transform
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
DJ	Disk Jockey
DNN	Deep Neural Network
DP	Dynamic Programming
DTW	Dynamic Time Warping
EBU	European Broadcasting Union
EM	Expectation Maximization
ERB	Equivalent Rectangular Bandwidth
FFT	Fast Fourier Transform
FT	Fourier Transform
FN	False Negative
FNR	False Negative Rate
FP	False Positive
FPR	False Positive Rate

FWR	Full-Wave Rectification
GMM	Gaussian Mixture Model
HFC	High Frequency Content
HMM	Hidden Markov Model
HPS	Harmonic Product Spectrum
HSS	Harmonic Sum Spectrum
HTK	HMM Toolkit
HWR	Half-Wave Rectification
IBI	Inter-Beat Interval
ICA	Independent Component Analysis
IDFT	Inverse Discrete Fourier Transform
IFT	Inverse Fourier Transform
IIR	Infinite Impulse Response
IO	Input/Output
IOI	Inter-Onset Interval
ITU	International Telecommunication Union
JNDL	Just Noticeable Difference in Level
K-NN	K-Nearest Neighbor
LDA	Linear Discriminant Analysis
MA	Moving Average
MAE	Mean Absolute Error
MFCC	Mel Frequency Cepstral Coefficient
MIDI	Musical Instrument Digital Interface
MIR	Music Information Retrieval
MIREX	Music Information Retrieval Evaluation eXchange
ML	Machine Learning
MPA	Music Performance Analysis
MP3	MPEG-1 Layer 3
MSE	Mean Squared Error
NOT	Note Onset Time
NMF	Nonnegative Matrix Factorization
PAT	Perceptual Attack Time
PCA	Principal Component Analysis
PDF	Probability Density Function
POT	Perceptual Onset Time
PPM	Peak Program Meter
PSD	Peak Structure Distance
RFD	Relative Frequency Distribution
RLB	Revised Low-Frequency B Curve
RMS	Root Mean Square
RNN	Recurrent Neural Network

ROC	Receiver Operating Characteristic Curve
SIMD	Single Instruction Multiple Data
SNR	Signal-to-Noise Ratio
SOM	Self-Organizing Map
SSM	Self Similarity Matrix
STFT	Short Time Fourier Transform
SVD	Singular Value Decomposition
SVM	Support Vector Machine
SVR	Support Vector Regression
TN	True Negative
TNR	True Negative Rate
TP	True Positive
TPR	True Positive Rate

List of Symbols

A	Amplitude
a	Filter Coefficient (Recursive)
Acc	Accuracy
$\mathcal{B}$	Number of Beats
b	Filter Coefficient (Transversal)
β	Parametrization Factor or Exponent
$\mathcal{C}$	Number of (Audio) Channels
$\chi(.)$	Center Clipping Function
$\boldsymbol{C}_\mathrm{AB}$	Cost Matrix for the Distance Matrix between Sequences A and B
$\mathfrak{C}_\mathrm{AB}$	Overall Cost of a Path through the Cost Matrix
$c_x(\cdot)$	Cepstrum of the Signal x
$\boldsymbol{D}_\mathrm{AB}$	Distance Matrix between Sequences A and B
d	Distance Measure
Δ_Q	Quantization Step Size
$\delta(\cdot)$	Delta Impulse Function
$\delta_\mathrm{T}(\cdot)$	Delta Pulse Function
$\mathfrak{d}$	Novelty Function
e_P	Prediction Error
e_Q	Quantization Error
$\mathfrak{e}(f)$	Equivalent Rectangular Bandwidth
η	(Correlation) Lag
F	F-Measure
f	Frequency in Hz
f_0	Fundamental Frequency in Hz
f_S	Sample Rate
f_A4	Tuning Frequency in Hz
$\mathcal{F}$	Number of Features
f_I	Instantaneous Frequency in Hz
$\mathfrak{F}(\cdot)$	(Discrete) Fourier Transform

G	Threshold
$\mathbf{\Gamma}$	Chord Transformation Matrix
$\gamma_{x,\mathcal{O}}$	Central Moment of Order $\mathcal{O}$ of Signal x
$H(\cdot)$	Transfer Function
$h(\cdot)$	Impulse Response
$\mathcal{H}$	Hop Size
i	Sample Index
$\mathcal{J}$	Impulse Response Length
j	Integer (Loop) Variable
J	Objective Function
$\mathcal{K}$	Block Size
k	Frequency Bin Index
κ	Percentage
λ	Weighting Factor
$\mathcal{M}$	Number of (Quantization) Steps
m	Slope
$\mathfrak{m}(f)$	Pitch (Mel)
Mg_x	Geometric Mean of Signal x
Mh_x	Harmonic Mean of Signal x
μ_x	Arithmetic Mean of Signal x
$\mathcal{N}$	Number of Observations or Blocks
n	Block Index
$\mathcal{O}$	Order (e.g. Filter Order)
o_r	Block Overlap Ratio
ω	Angular Velocity ($\omega = 2\pi f$) in radians per second
O	Number of Onsets
P	Precision
P	Alignment Path
PC	Pitch Class Index
Φ_X	Phase Spectrum of the Signal x
$\varphi(\cdot)$	Gaussian Function
$\mathfrak{p}$	(MIDI) Pitch
ν	Pitch Chroma Vector/Key Profile
P_x	Power of the Signal x
$p_x(x)$	Probability Density Function of the Signal x
$\psi(\cdot)$	Chord Probability Vector
$\mathcal{Q}$	Quality Factor (Mid-Frequency divided by Bandwidth)
q	Evaluation Metric
$Q_x(\cdot)$	Quantile Boundary
R	Recall
$r_{xy}(\cdot)$	Correlation Function between the Signals x and y

r	Radius
$\boldsymbol{R}$	Covariance Matrix
$\boldsymbol{S}$	Similarity Matrix
σ_x	Standard Deviation of Signal x
σ_x^2	Variance of Signal x
SNR	Signal-to-Noise Ratio
T	Time Period in s
t	Time in s
T_0	Time Period of the Fundamental Frequency in s
$\mathcal{T}$	Number of (Chord) Templates
$\mathfrak{T}$	Tempo in BPM
$\boldsymbol{T}$	(PCA) Transformation Matrix
T_S	Sample Period in s
$\boldsymbol{V}$	Feature Matrix with dimensions $\mathcal{F} \times \mathcal{N}$
v_ACF^η	ηth Autocorrelation Coefficient
v_C	Centroid
$\mathcal{V}$	Feature Set
v_K	Kurtosis
v_MFCC^j	jth MFCC
v_Peak	Peak Envelope
v_PPM	Peak Program Meter
v_RMS	RMS
v_SC	Spectral Centroid
v_SD	Spectral Decrease
v_SF	Spectral Flux
v_SK	Spectral Kurtosis
v_Sk	Skewness
v_SR	Spectral Rolloff
v_SS	Spectral Spread
v_SSk	Spectral Skewness
v_SSl	Spectral Slope
v_Ta	ACF Maximum
v_Tf	Spectral Flatness
v_Tpr	Tonal Power Ratio
v_Tsc	Spectral Crest Factor
v_ZC	Zero Crossing Rate
w	Word Length in Bit
w_AB	Window Function with Alternative Blackman Shape
w_B	Window Function with Blackman Shape
w_BH	Window Function with Blackman–Harris Shape
w_C	Window Function with Cosine Shape

w_{H}	Window Function with von-Hann Shape
w_{Hm}	Window Function with Hamming Shape
w_{R}	Window Function with Rectangular Shape
w_{T}	Window Function with Bartlett Shape
$X(\cdot)$	Fourier Representation of the Signal x
$\mathfrak{x}(f)$	Normed Frequency Position on the Cochlea
$X^*(\cdot)$	Conjugate-Complex Spectrum of the Signal x
$\mathfrak{z}(f)$	Critical Band Rate (Bark)

Source Code Repositories

Book reference	**Link**	**Description**
Repo 1	https://github.com/alexanderlerch/ACA-Plots	Matlab code to generate plots
Repo 2	https://github.com/alexanderlerch/ACA-Code	Matlab scripts
Repo 3	https://github.com/alexanderlerch/pyACA	Python scripts
Repo 4	https://github.com/alexanderlerch/ACA-Slides	Pdf and LaTeX code for lecture slides

1

Introduction

Audio is an integral and ubiquitous aspect of our daily lives; we intentionally produce sound (e.g. when communicating through speech or playing an instrument), we actively listen (e.g. to music or podcasts), can focus on a specific sound source in a mixture of sources, and we (even unconsciously) suppress sound sources internally (e.g. traffic noise). Similar to humans, algorithms can also generate, analyze, and process audio. This book focuses on the algorithmic analysis of audio signals, more specifically the extraction of information from musical audio signals.

Audio signals contain a wealth of information: by simply listening to an audio signal, humans are able to infer a variety of content information. A speech signal, for example obviously transports the textual information, but it also might reveal information about the speaker (gender, age, accent, mood, etc.), the recording environment (e.g., indoors vs. outdoors), and much more. A music signal might allow us to derive melodic and harmonic characteristics, understand the musical structure, identify the instruments playing, perceive the projected emotion, categorize the music genre, and assess characteristics of the performance as well as the proficiency of the performers. An audio signal can contain and transport a wide variety of content beyond these simple examples. This content information is sometimes referred to as *metadata*: data about (audio) data.

The field of *Audio Content Analysis (ACA)* aims at designing and applying algorithms for the automatic extraction of content information from the raw (digital) audio signal. This enables content-driven and content-adaptive services which describe, categorize, sort, retrieve, segment, process, and visualize the signal and its content.

The wide range of possible audio sources and the multi-faceted nature of audio signals results in variety of distinct ACA problems, leading to various areas of research, including

- *speech analysis*, covering topics such as automatic speech recognition [1, 2] or recognizing emotion in speech [3, 4],

An Introduction to Audio Content Analysis: Music Information Retrieval Tasks & Applications,
Second Edition. Alexander Lerch.
© 2023 The Institute of Electrical and Electronics Engineers, Inc. Published 2023 by John Wiley & Sons, Inc.

- *urban sound analysis* with applications in noise pollution monitoring [5] and audio surveillance, i.e. the detection of dangerous events [6],
- *industrial sound analysis* such as monitoring the state of mechanical devices like engines [7] or monitoring the health of livestock [8], and, last but not least,
- *musical audio analysis*, targeting the understanding and extraction of musical parameters and properties from the audio signal [9].

This book focuses on the analysis of musical audio signals and the extraction of musical content from audio. There are many similarities and parallels to the areas above, but there exist also many differences that distinguish musical audio from other signals beyond simple technical properties such as audio bandwidth. Like an urban sound signal, music is a polytimbral mixture of multiple sound sources, but unlike urban sound, its sound sources are clearly related (e.g., melodically, harmonically, or rhythmically). Like a speech signal, a music signal is a sequence in a language with rules and constraints, but unlike speech, the musical language is abstract and has no singular meaning. Like an industrial audio signal, music has both tonal and noise-like components which may repeat themselves, but unlike the industrial signal, it conveys a (musical) form based on hierarchical grouping of elements not only through repetition but also through precise variation of rhythmic, dynamic, tonal, and timbral elements.

As we will see throughout the chapters, the design of systems for the analysis of musical audio often requires knowledge and understanding from multiple disciplines. While this text approaches the topic mostly from an engineering and Digital Signal Processing (DSP) perspective, the proper formulation of research questions and task definitions often require methods or at least synthesis of knowledge from fields as diverse as music theory, music perception, and psychoacoustics. Researchers working on ACA thus come from various backgrounds such as computer science, engineering, psychology, and musicology.

The diversity in musical audio analysis is also exemplified by the wide variety of terms referring to it. Overall, musical ACA is situated in the broader area of *Music Information Retrieval (MIR)*. MIR is a broader field that covers not only the analysis of musical audio but also symbolic (nonaudio) music formats such as musical scores and files or signals compliant to the so-called *Musical Instrument Digital Interface (MIDI)* protocol [10]. MIR also covers the analysis and retrieval of information that is music-related but cannot be (easily) extracted from the audio signal such as the artist names, user ratings, performance instructions in the score, or bibliographical information such as publisher, publishing date, the work's title. Other areas of research, such as music source separation and automatic music generation are often also considered to belong within MIR. Various overview articles clarify how the understanding of the field of MIR has evolved over time [11–16]. Other, related terms are also in use. Audio event detection, nowadays often related to urban sound analysis, is sometimes described as computational analysis of sound scenes [17]. The analysis of sound scenes from a perceptual point of view has been described as *Computational Auditory Scene Analysis (CASA)* [18]. In the past, other terms have been used more or less synonymously to the term "audio content" analysis. Examples of such synonyms are *machine listening* and *computer audition*. Finally, there is the term *music informatics*, which encompasses essentially any aspect of algorithmic analysis, synthesis, and processing of music (although in some circles, its meaning is restricted to describe the creation of musical artifacts with software).

1.1 A Short History of Audio Content Analysis

Historically, the first systems analyzing the content of audio signals appeared shortly after technology provided the means of storing and reproducing recordings on media in the twentieth century. One early example is Seashore's Tonoscope, which enabled the pitch analysis of an audio signal by visualizing the fundamental frequency of the incoming audio signal on a rotating drum [19]. However, the more recent evolution of digital storage media, DSP methods, and machine learning during the last decades, along with the growing amount of digital audio data available through downloads and streaming services, has significantly increased both the need and the possibilities of automatic systems for analyzing audio content, resulting in a lively and growing research field.

Early systems for audio analysis were frequently so-called "expert systems" [20], designed by experts who implement their task-specific knowledge into a set of rules. Such systems can be very successful if there is a clear and simple relation between the knowledge and the implemented algorithms. A good example for such systems are some of the pitch-tracking approaches introduced in Section 7.3: as the goal is the detection of periodicity in the signal in a specific range, an approach such as the Autocorrelation Function (ACF), combined with multiple assumptions and constraints, can be used to estimate this periodicity and thus the fundamental frequency.

Later, data-driven systems became increasingly popular and traditional machine learning approaches started to show superior performance on many tasks. These systems extract so-called features from the audio to achieve a task-dependent representation of the signal. Then, training data are used to build a model of the feature space and how it maps to the inferred outcome. The role of the expert becomes less influential in the design of these systems, as they are restricted to selecting or designing a fitting set of features, curating a representative set of data, and choosing and parametrizing the machine learning approach. One prototypical example for these approaches is musical genre classification as introduced in Section 12.2.

Modern machine learning approaches include trainable feature extraction approaches (also referred to as feature learning, see Section 3.6) as they are a part of deep neural networks. These approaches have consistently shown superior performance for nearly all ACA tasks. The researcher seldom imparts much domain knowledge beyond choosing input representation, data, and system architecture of these systems. An example of such an end-to-end system could be a music genre classification system based on a neural network with a convolutional architecture with a Mel Spectrogram input.

It should be pointed out that while modern systems tend to have superior performance, they also tend to be less interpretable and explainable than traditional systems. For example, deducing the reason for a false classification result in a network-based system can be difficult, while the reason is usually easily identifiable in a rule-based system.

1.2 Applications and Use Cases

The content extracted from music signals improves or enables various forms of content-based and content-adaptive services, which allow to sort, categorize, find, segment, process, and visualize the audio signal based on its content.

1.2.1 Music Browsing and Music Discovery

One of the most intuitive applications is the content-based search for music in large databases for which consistent manual annotation by humans is often infeasible. This allows, for example, Disk Jockeys (DJs) to retrieve songs in a specific tempo or key, to find music pieces in a specific genre or mood, or to search for specific musical characteristics by example. The same information can be used in end consumer applications such as audio-based music recommendation and playlist generation systems using an in-depth understanding of the musical content [21].

The content-based annotation and representation of the audio signal can also be used to design new interfaces to access data. Fingerprinting (see Chapter 11), for example allows to identify a song currently played from a large database of songs, while Query-by-Humming systems identify songs through a user-hummed melody. Content can also be utilized for new ways of sound visualization and user interaction; a database of music could, for example be explored by a user navigating a virtual music similarity space [22].

1.2.2 Music Consumption

Audio analysis has already started to transform consumer-facing industries such as streaming services as mentioned above. So far, most services focus on recommending music to be played back, but in the near future, we might see the rise of creative music listening applications that enable the listener to interact with the content itself instead of being restricted to only choosing content to be played. This could include, for example, the gain adjustment for individual voices, replacing instruments or vocalists, or interactively changing the musical arrangement or even stylistic properties of the music piece.

1.2.3 Music Production

Knowledge of the audio content can improve music production tools in various dimensions. On the one hand, content information can enable a more "musical" software interface, e.g. by displaying score-like information synchronized with the audio data, and thus enabling an intuitive approach to editing the audio data. On the other hand, production software usage could be enhanced in terms of productivity and efficiency: the better a system understands the details and properties of incoming audio streams or files, the better it can adapt, for instance, by applying default gain and equalization parameters [23] or by suggesting compatible audio from a sound library. Intelligent music software might also support editors by offering automatic artifact-free splicing of multiple recordings from one session or selecting error-free recordings from a set of recordings.

Modern tools also enhance the creative possibilities in the production process. For example, creating harmonically meaningful background choirs by analyzing the lead vocals and the harmony track is already technically feasible and commercially available today. Knowing and isolating sound sources in a recording could enable new ways of modifying or morphing different sounds for the creation of new soundscapes, effects, and auditory scenes.

1.2.4 Music Education

The potential of utilizing technology to assist music (instrument) education has been recognized as early as the 1930s, when Seashore pointed out the educational value of scientific observation of music performances [24]. The availability of automatic and fast audio analysis can support the creation of artificially intelligent music tutoring software, which aims at supplementing teachers by providing students with insights and interactive feedback by analyzing and assessing the audio of practice sessions. An interactive music tutor with in-depth understanding of the musical score and performance content can highlight problematic parts of the students' performance, provide a concise yet easily understandable analysis, give objective and specific feedback on how to improve, and individualize the curriculum depending on the students' mistakes and general progress. At the same time, music tutoring software can provide an accessible, objective, and reproducible analysis.

1.2.5 Generative Music

Machine-interpretable content information can also feed generative algorithms. The automatic composition and rendition of music is emerging as a challenging yet popular research direction [25], gaining interest from both research institutions and industry. While bigger questions concerning capabilities and restrictions of computational creativity as well as aesthetic evaluation of algorithmically generated music remain largely unanswered, practical applications such as generating background music for user videos and commercial advertisements are currently in the focus of many researchers. The interactive and adaptive generation of sound tracks for video games as well as individualized generation of license-free music content for streaming is additional long-term goals of considerable commercial interest.

References

1 Xuedong Huang, James Baker, and Raj Reddy. A Historical Perspective of Speech Recognition. *Communications of the ACM*, 57(1):94–103, January 2014. ISSN 0001-0782. doi: 10.1145/2500887. URL https://doi.org/10.1145/2500887.

2 Dong Yu and Li Deng. *Automatic Speech Recognition - A Deep Learning Approach. Signals and Communication Technology*. Springer, London, 2015. ISBN 978-1-4471-5779-3. URL https://doi.org/10.1007/978-1-4471-5779-3.

3 Frank Dellaert, Thomas Polzin, and Alex Waibel. Recognizing Emotion in Speech. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, volume 3, pages 1970–1973, Philadelphia, PA, October 1996. doi: 10.1109/ICSLP.1996.608022.

4 Moataz El Ayadi, Mohamed S Kamel, and Fakhri Karray. Survey on Speech Emotion Recognition: Features, Classification Schemes, and Databases. *Pattern Recognition*, 44(3):572–587, March 2011. ISSN 0031-3203. doi: 10.1016/j.patcog.2010.09.020. URL https://www.sciencedirect.com/science/article/pii/S0031320310004619.

5 Juan Pablo Bello, Charlie Mydlarz, and Justin Salamon. Sound Analysis in Smart Cities. In *Computational Analysis of Sound Scenes and Events*, pages 373–397. Springer, Cham, 2018. ISBN 978-3-319-63449-4. URL https://link.springer.com/chapter/10.1007/978-3-319-63450-0_13.

6 Marco Crocco, Marco Cristani, Andrea Trucco, and Vittorio Murino. Audio Surveillance: A Systematic Review. *ACM Computing Surveys*, 48(4):52:1–52:46, February 2016. ISSN 0360-0300. doi: 10.1145/2871183. URL https://doi.org/10.1145/2871183.

7 Sascha Grollmisch, Jakob Abeßer, Judith Liebetrau, and Hanna Lukashevich. Sounding Industry: Challenges and Datasets for Industrial Sound Analysis.

In *Proceedings of the European Signal Processing Conference (EUSIPCO)*, pages 1–5, A Coruna, Spain, September 2019. doi: 10.23919/EUSIPCO.2019.8902941. ISSN: 2076-1465.

8 Dries Berckmans, Martijn Hemeryck, Daniel Berckmans, Erik Vranken, and Toon van Waterschoot. Animal Sound... Talks! Real-time Sound Analysis for Health Monitoring in Livestock. In *Proceedings of the International Symposium on Animal Environment and Welfare (ISAEW)*, Chongqing, China, 2015.

9 Daniel P W Ellis. Extracting Information from Music Audio. *Communications of the ACM*, 49(8):32–37, August 2006. ISSN 00010782. doi: 10.1145/1145287.1145310. URL http://portal.acm.org/citation.cfm?doid=1145287.1145310.

10 MIDI Manufacturers Association. Complete MIDI 1.0 Detailed Specification V96.1, 2nd edition. Standard, MMA, 2001.

11 J Stephen Downie. Music Information Retrieval. *Annual Review of Information Science and Technology*, 37:295–340, 2003.

12 Nicola Orio. Music Retrieval: A Tutorial and Review. *Foundations and Trends in Information Retrieval*, 1(1):1–90, 2006.

13 Michael A Casey, Remco Veltkamp, Masataka Goto, Marc Leman, Christophe Rhodes, and Malcolm Slaney. Content-based Music Information Retrieval: Current Directions and Future Challenges. *Proceedings of the IEEE*, 96(4):668–696, April 2008. ISSN 1558-2256. doi: 10.1109/JPROC.2008.916370. Conference Name: Proceedings of the IEEE.

14 Markus Schedl, Emilia Gómez, and Julián Urbano. Music Information Retrieval: Recent Developments and Applications. *Foundations and Trends® in Information Retrieval*, 8(2-3):127–261, September 2014. ISSN 1554-0669, 1554-0677. doi: 10.1561/1500000042. URL http://www.nowpublishers.com/article/Details/INR-042.

15 Alexander Lerch. Music Information Retrieval. In Stefan Weinzierl, editor, *Akustische Grundlagen der Musik*, number 5 in Handbuch der Systematischen Musikwissenschaft, pages 79–102. Laaber, 2014. ISBN 978-3-89007-699-7.

16 John Ashley Burgoyne, Ichiro Fujinaga, and J Stephen Downie. Music Information Retrieval. In Susan Schreibman, Ray Siemens, and John Unsworth, editors, *A New Companion to Digital Humanities*, pages 213–228. John Wiley & Sons, Ltd, 2015. ISBN 978-1-118-68060-5. doi: 10.1002/9781118680605.ch15. URL http://onlinelibrary.wiley.com/doi/10.1002/9781118680605.ch15/summary.

17 Tuomas Virtanen, Mark D Plumbley, and Dan Ellis, editors. *Computational Analysis of Sound Scenes and Events*. Springer International Publishing, Cham, 2018. ISBN 978-3-319-63449-4 978-3-319-63450-0. doi: 10.1007/978-3-319-63450-0. URL http://link.springer.com/10.1007/978-3-319-63450-0.

18 DeLiang Wang and Guy J Brown. *Computational Auditory Scene Analysis: Principles, Algorithms, and Applications*. Wiley-IEEE Press, 2006. ISBN 978-0-470-04338-7. URL https://ieeexplore.ieee.org/book/5769523.

19 Carl E Seashore. A Voice Tonoscope. *Studies in Psychology*, 3:18–28, 1902.

20 Peter J F Lucas and Linda C van der Gaag. *Principles of Expert Systems*. Addison-Wesley, 1991.

21 Peter Knees, Markus Schedl, and Masataka Goto. Intelligent User Interfaces for Music Discovery: The Past 20 Years and What's to Come. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 44–53, Delft, Netherlands, 2019. URL http://archives.ismir.net/ismir2019/paper/000003.pdf.

22 Masahiro Hamasaki, Masataka Goto, and Tomoyasu Nakano. Songrium: Browsing and Listening Environment for Music Content Creation Community. In *Proceedings of the Sound and Music Computing Conference (SMC)*, page 8, Maynooth, Ireland, 2015.

23 Joshua D Reiss and Øyvind Brandtsegg. Applications of Cross-Adaptive Audio Effects: Automatic Mixing, Live Performance and Everything in Between. *Frontiers in Digital Humanities*, 5, 2018. ISSN 2297-2668. doi: 10.3389/fdigh.2018.00017. URL https://www.frontiersin.org/articles/10.3389/fdigh.2018.00017/full.

24 Carl E Seashore. *Psychology of Music*. McGraw-Hill, New York, 1938.

25 Jean-Pierre Briot, Gaëtan Hadjeres, and François-David Pachet. *Deep Learning Techniques for Music Generation*. Computational Synthesis and Creative Systems. Springer International Publishing, Cham, 2020. ISBN 978-3-319-70162-2 978-3-319-70163-9. doi: 10.1007/978-3-319-70163-9. URL http://link.springer.com/10.1007/978-3-319-70163-9.

Part I

Fundamentals of Audio Content Analysis

2

Analysis of Audio Signals

Taking a closer look at Audio Content Analysis (ACA), this chapter introduces the musical content to be extracted from the audio signal and then summarizes the basic processing steps of an ACA system.

2.1 Audio Content

The introduction already provided some examples for content that can be extracted from an audio signal. For a recording of Western music, content originates from three different sources:

- *Composition*: The term composition will be used broadly as a definition and conceptualization of musical ideas.[1] It can refer to any form of musical notation or description. This includes traditional ways from the *basso continuo* (a historic way of defining the harmonic structure) to the classic Western score notation as well as the modern lead sheet and other forms of notation used for contemporary and popular music.

  The content related to the composition allows us to recognize different renderings of the same song or symphony as being the same piece. In most genres of Western music, this can encompass musical elements such as melody, harmony, instrumentation, structure and form, and rhythm.
- *Performance*: Music generally requires a performer or group of performers to create a unique acoustical rendition of the composition. The performance not only communicates the explicit information from the composition but also interprets and modifies it.

  This happens through the performance-related content, which includes, for example, the tempo and its variation as well as the micro-timing, the realization

---

[1] In Chapter 15, we will also refer to it as a *blueprint* for a performance.

An Introduction to Audio Content Analysis: Music Information Retrieval Tasks & Applications,
Second Edition. Alexander Lerch.
© 2023 The Institute of Electrical and Electronics Engineers, Inc. Published 2023 by John Wiley & Sons, Inc.

of musical dynamics, accents, and instantaneous dynamic modulations such as tremolo, the use of expressive intonation and vibrato, and specific playing (e.g. bowing) techniques influencing the sound quality.

- *Production*: As the input of an ACA system is an audio recording, the choices made during recording and production impact certain characteristics of the recording.
  Production-related content mostly impacts the sound quality of the recording (microphone positioning, equalization, and the application of effects such as reverb to the signal) and the dynamics (by applying manual or automatic gain adjustments). However, changes in timing and pitch may occur as well during the process of editing the recording and applying software for pitch correction.

Note that these content sources can be hard to separate from the final audio recording. For example, the timbre of a recording can be determined by the instrumentation indicated by the score, by the performers' choice of instruments (e.g. historical instruments, specific guitar amps), by specific playing techniques, and by sound-processing choices made by the sound engineer or the producer.

Furthermore, it should be noted that the distinction between composition and performance becomes less clearly defined in improvisatory and non-Western forms of music.

While helpful for a better understanding of musical audio, the categorization by content source has only limited use from a practical audio analysis point of view. A more useful categorization of musical content is driven by musical and perceptual considerations, leading to four main categories:

- *timbre characteristics*: information related to sound quality such as spectral shape (see Section 3.5),
- *tonal characteristics*: pitch-related information such as melody, harmony, key, intonation (see Section 7),
- *intensity-related characteristics*: dynamics-related information such as envelope-, level-, and loudness (see Section 8), and
- *temporal characteristics*: information comprising the tempo, timing, meter, of the signal (see Section 9).

Obviously, there are additional ways to quantitatively describe audio signals that do not fall squarely in one of these music-driven categories, for example, statistical or other technical descriptions of the audio data such as an amplitude distribution.

The categories listed above cover the basic musical information that can be found in a music signal, and analysis tasks such as tempo detection, key detection, or envelope extraction fall unambiguously into one of the above categories. However, cues from various categories are necessary to deduce, for example, the structure of a musical piece or to extract performance characteristics.

Furthermore, there exist many high-level perceptual concepts that humans use when categorizing music, such as the music genre or the projected affective content. Such concepts draw from a combination of several or all categories mentioned above, albeit the impact of specific musical parameters on the perception of these concepts is hard to quantify.

Note that there is also content that is not easily extractable from the signal itself but may still be useful for music information retrieval (MIR) systems. Examples for this content include, for example, the year of the composition or recording, the record label, the song title, and information on the artists.

2.2 Audio Content Analysis Process

Systems for audio content analysis can often be represented as a two-step process as depicted in Figure 2.1, where the signal is first converted into a meaningful input or feature representation that is then fed into a rule-based or data-driven system for inference such as a classifier. A perfect, maximally meaningful, input representation requires only a trivial inference method: inference could be as simple as just applying a threshold to the representation. A very powerful method of inference, on the other hand, is able to utilize an input representation even if the task-relevant information is not directly or easily accessible from the representation. In traditional systems, the design of the input representation used to require considerable effort in feature design and engineering. Nowadays, the input processing tends to become less sophisticated (e.g. simply computing a spectrogram), which is counterbalanced by the increasing complexity of the inference system.

A preprocessing step can precede the feature extraction stage; in this step, the input data might be converted into a specific format, normalized, or processed for increased robustness and stability.

For traditional feature-based approaches, the following properties are generally desirable for the feature input:

- *Task-relevance without noise or other task-irrelevant information*: features necessarily have to represent information that is relevant to the task at hand. A tempo detection system, for example, might need to focus on the rhythmic

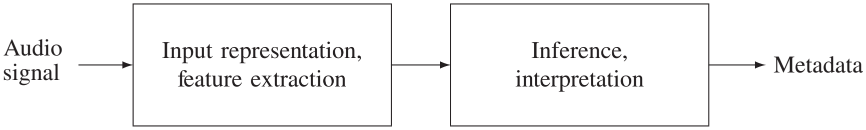

Audio signal
Input representation, feature extraction
Inference, interpretation
Metadata

Figure 2.1 General processing stages of a system for audio content analysis.

parts instead of pitch-related information. Therefore, a meaningful feature captures task-relevant (rhythmic) information and discards task-irrelevant (pitch-related) information. Note that capturing task-relevant information does not necessarily mean that such features are humanly interpretable; rather, it is sufficient if they can be properly interpreted by the inference system.

- *Compact and nonredundant*: the dimensionality of raw audio data tends to be too high for traditional machine learning approaches. One channel of a digital audio file in Compact Disc (CD) quality (44 100 samples per second, 16 Bits per sample[2]) with a length of five minutes contains

  $$5\text{ minutes} \cdot 60\ {}^{\text{s}}\!/_{\text{min}} \cdot\ 44100\ {}^{\text{samples}}\!/_{\text{s}} \cdot 16\ {}^{\text{bits}}\!/_{\text{sample}} = 211{,}680{,}000\text{ bits} \qquad (2.1)$$

  A compact feature aims at representing these data with fewer values.
- *Easy to analyze*: Although all extractable information is contained in the raw audio data, observing meaningful information directly from the time-domain audio data is complicated. For example, a spectrogram contains a similar amount of data as a time domain signal, but it is often easier to parse due to its more sparse representation of the content.

While these qualities continue to be desirable for modern approaches based on neural networks, the features are not explicitly designed but rather learned from data. The input of the machine learning system is, thus, not a custom-designed representation but the more or less unprocessed raw data. Some argue that end-to-end systems without input processing are generally preferable [1], however, this statement is debatable [2]. It should be noted that the terms input representation and feature representation are often used inconsistently. For example, representation learning aims at extracting a compressed feature representation from data that can be potentially used for a multitude of related tasks. This representation might be the activation at one of the deeper layers of a neural network; these activations, also referred to as embeddings, might then be referred to as features which might in turn be at the input of another network.

In a way, this ambiguity relates to the various abstraction levels between the raw audio data and how humans refer to music. While extracting the tempo might be the output of one system, another system might use this tempo just as one feature among many others to, for example, automatically recognize the musical style.

The second processing stage of an ACA system infers the desired information from the feature data. Thus, it aims at mapping the potentially abstract feature information into a domain both usable and comprehensible by humans. This process is often accomplished with a data-driven machine learning system, but might also be a rule-based expert system.

---

[2] Note that an audio sample is usually represented with a 32 Bit float in music processing and analysis software.

2.3 Exercises

2.3.1 Questions

1. Describe the differences between score-inherent and performance-inherent musical content in the context of Western classical music.
2. Discuss why the boundaries between score and performance are less clearly defined for improvisatory forms of music.
3. List 10 examples of content in a musical audio signal. Assign one (or more) of the four basic content categories to each of your examples.

References

1 Jordi Pons, Oriol Nieto, Matthew Prockup, Erik M Schmidt, Andreas F Ehmann, and Xavier Serra. End-to-end Learning for Music Audio Tagging at Scale. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Paris, France, September 2018. URL https://zenodo.org/record/1492497#.YBb323dKg8M.

2 Tobias Glasmachers. Limits of End-to-End Learning. In *Asian Conference on Machine Learning*, pages 17–32. PMLR, November 2017. URL http://proceedings.mlr.press/v77/glasmachers17a.html. ISSN: 2640-3498.

3

Input Representation

This chapter introduces typical input representations of an Audio Content Analysis (ACA) system and describes their computation. First, it covers some fundamental properties of audio signals and the description of audio signals. Then, it outlines typical preprocessing steps such as blocking and down-mixing. Then, time-frequency representations of the audio signal are introduced, followed by the description of often-used features and their postprocessing.

3.1 Audio Signals

An *audio signal* as perceived by humans can be described as a function of time-variant sound pressure level. A microphone can be used to convert the sound pressure level into voltage. The signal is defined for all times t and is therefore a (time-) continuous signal $x(t)$.

Similar to an image, all signal content of a digital audio signal is encoded through numerical values; however, despite many analogies between audio and image signals, there are some key points that differentiate an audio signal from an image. First, an audio signal is a time series; most content is conveyed over the course of time and can change over time. Second, multiple objects in an audio signal are superposed; although loud sources can completely or partially mask quieter ones, they cannot completely suppress the other signal like a large object can block the view of a smaller object in an image. Third, most music signals have two or more audio channels to provide spatial information; while various forms of stereoscopy exist for images, they are not as common as for music signals. Nevertheless, there are a fair number of audio processing approaches inspired by or similar to image processing methods, some of which will be introduced later in the text.

It is common to describe a typical audio signal as a time-variant mixture of tonal, pitched components, and noise components. Ignoring the time-variance, it is

An Introduction to Audio Content Analysis: Music Information Retrieval Tasks & Applications,
Second Edition. Alexander Lerch.
© 2023 The Institute of Electrical and Electronics Engineers, Inc. Published 2023 by John Wiley & Sons, Inc.

common to model pitched components as periodic signals and noisy components as random signals.

3.1.1 Periodic Signals

A *periodic signal* repeats itself in constant time intervals. Its periodicity can be formulated by

$$x(t) = x(t + T_0) \tag{3.1}$$

with T_0 being the periodicity interval, or, in other words, the *period length* of the lowest frequency (i.e. the first of the harmonics). Figure 3.1 shows a periodic audio signal and highlights its fundamental period length T_0.

The *fundamental frequency* of this periodic signal is then

$$f_0 = \frac{1}{T_0}. \tag{3.2}$$

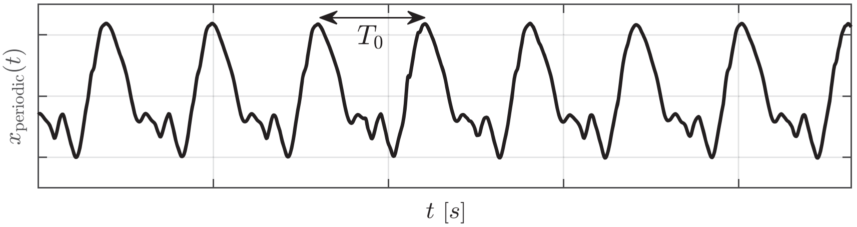
The frequency of other tonal components, the remaining harmonics, is – for periodic signals – always an integer multiple of the frequency of the first harmonic f_0.

Every periodic signal $x(t)$ can be represented by a superposition of weighted sinusoidal signals, the *Fourier series*

$$x(t) = \sum_{k=-\infty}^{\infty} A(k) e^{\mathrm{j}\omega_0 kt}. \tag{3.3}$$

The periodicity, or the fundamental frequency, of the signal is represented as angular frequency $\omega_0 = 2\pi f_0$. The real part of $e^{\mathrm{j}\omega_0 kt}$ represents the cosine components (even) and its imaginary part the sine components (odd) of the signal: $e^{\mathrm{j}\omega t} = \cos(\omega t) + \mathrm{j}\sin(\omega t)$. The coefficient $A(k)$ is the weight of the kth harmonic and can be calculated by

$$A(k) = \frac{1}{T_0} \int\limits_{-{}^{T_0}\!/_2}^{{}^{T_0}\!/_2} x(t) e^{-\mathrm{j}\omega_0 kt}\, dt. \tag{3.4}$$



Repo 1: plotPeriodic.m

Figure 3.1 Snippet of a periodic audio signal with indication of its fundamental period length T_0.

If the number of frequencies used to represent the signal is limited, then some periodic signals may be constructed only imperfectly. The larger the order $\mathcal{O}$, the higher is the highest frequency included and the better is the approximation of the signal

$$\hat{x}(t) = \sum_{k=-\mathcal{O}}^{\mathcal{O}} A(k)e^{\mathrm{j}\omega_0 kt}. \tag{3.5}$$

Periodic signals featuring "sudden changes" or discontinuities of the waveform require higher (or infinite)-order Fourier coefficients to model the waveform sufficiently well. This also means that signals with those sudden changes, e.g. transients, contain a considerable amount of high-frequency content. More systematically, we can say that the higher the order of derivatives of the signal that are monotone is, the lower the required order will be to represent the signal sufficiently well. The most extreme example is a sinusoidal signal which has an infinite number of monotone derivatives and for which only one coefficient $A(1)$ is required to represent the signal perfectly.

Figure 3.2 shows on the left one period of two periodic signals, a *sawtooth* (top) and a *square wave* (bottom), modeled with orders 3 and 50. The coefficients $A(k)$ for these two signals, shown on the right of the figure, are

$$A_{\mathrm{saw}}(k) = \frac{2}{\pi \cdot k}, \tag{3.6}$$

$$A_{\mathrm{rect}}(k) = \frac{4}{\pi \cdot (2k-1)}. \tag{3.7}$$

Due to the discontinuities of these two signals there are model errors in the form of overshoots around the point of discontinuity. It can be observed that the frequency of the overshoots increases and the duration of the overshoots decreases

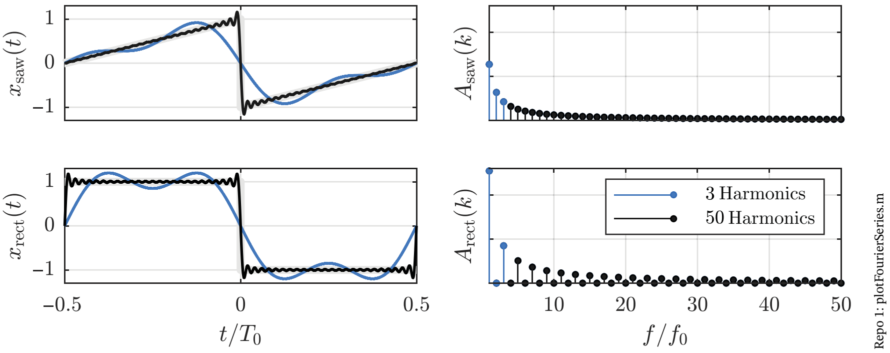


Figure 3.2 Approximation of periodic signals: sawtooth (top) and square wave (bottom) reconstructed with 3 and 50 sinusoidal harmonics; left: time domain signal, right: Fourier coefficients.

with increasing model order. However, the amplitude of the overshoots stays constant. This is called *Gibbs' phenomenon* [1, 2].

3.1.2 Random Signals

In contrast to periodic signals, future values of *random signals* cannot be predicted no matter how long the signal has been observed. That means that there exists no periodicity and thus no fundamental frequency for a random signal. Every observation of such a signal is therefore only an example of an unlimited number of possible incarnations of the signal. A complete signal description is theoretically only possible with an unlimited number of observations.

An important subcategory of random signals is built of *stationary* signals for which basic properties (such as arithmetic mean and higher central moments, see below) do not change over time. *White noise* is a typical example of a stationary random signal.

3.1.3 Statistical Signal Description

In contrast to, e.g. sinusoidal signals, noise cannot be described in the time domain with analytical functions. Statistics can help to identify some properties that describe the signal without defining specific amplitude values at a time. One of the most common representations is the *Probability Density Function (PDF)*, which is a useful tool to describe stationary processes or signals that have the same properties for any point in time t.

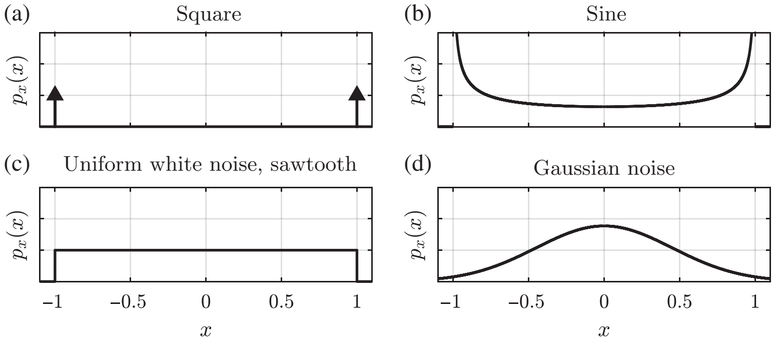
The PDF $p_x(x)$ of a signal is the probability distribution of a signal. The x-axis of a PDF plot represents all possible (amplitude) values of the signal x and their probability density is plotted on the y-axis. For example, a rectangular PDF means that all signal amplitudes between the maximum and minimum amplitudes occur with the same probability. The properties of the PDF are

$$p_x(x) \geq 0, \text{ and} \tag{3.8}$$

$$\int_{-\infty}^{\infty} p_x(x)\, dx = 1, \tag{3.9}$$

because the probability density can never be negative, and the probability of any possible amplitude occurring has to be 1.

A set of prototypical PDFs is shown in Figure 3.3. For a full-scale square wave, the PDF has only two peaks at maximum and minimum amplitudes, while other amplitude values do not exist. A sine wave has a PDF in which values similar to the arithmetic mean (here: 0) are less frequent than values far from the mean. A uniform PDF can be found with noise generated with the `rand` function of different programming languages; a sawtooth has such a uniform PDF as well



Repo 1: plotPdfExamples.m

Figure 3.3 Probability density function of a square wave (a), a sinusoidal (b), uniform white noise or a sawtooth wave (c), and Gaussian noise (d).

(which means that the PDF only captures certain aspects, not all properties of the signal). A *Gaussian distribution*

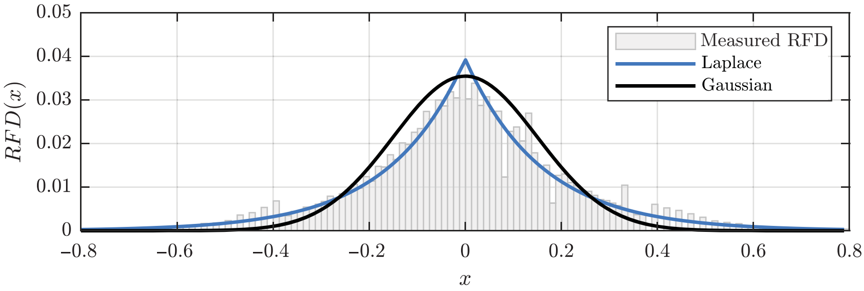
$$p_g = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu_x)^2}{2\sigma^2}\right) \tag{3.10}$$

is often assumed to be a typical distribution of "real-world" noise; a typical music signal has in many cases a distribution shaped similar to a *Laplace distribution* with a prominent peak at 0 amplitude (see Figure 3.4). The Laplace distribution is given by

$$p_l = \frac{1}{2\beta} \exp\left(-\frac{|x-\mu_x|}{\beta}\right) \tag{3.11}$$

with β as the scale parameter. If the input signal is a constant, for example a DC offset, the PDF consists only of a single peak.

Comparing Figures 3.3b and 3.4 makes it obvious that under the assumption that a PDF describes a signal sufficiently well, a sine wave – while a basic component



Repo 1: plotRfd.m

Figure 3.4 Distribution function estimated from a music signal compared to a Laplace and a Gaussian distribution.

of every pitched sound – is not a very good model of a real-world music or speech signal in terms of its distribution.

The PDF of quantized input signals has a limited set of amplitude classes as a quantized signal has only a limited set of amplitude values.

The PDF of a signal can be estimated from a sufficiently long block of samples by computing a histogram of signal amplitudes and dividing it by the number of observed samples. It is then sometimes referred to as *Relative Frequency Distribution (RFD)*. For the sake of simplicity, the PDF and the RFD will not be differentiated in this text.

Describing a signal statistically does not only make sense for audio signals, but it can also reveal important properties of other signals such as a series of features. For example, in a two-class problem, the feature distribution for one class might be very different from the feature distribution for the other class. A classifier can model these class distributions to allow distinguishing between classes.

3.1.3.1 Arithmetic Mean

The *arithmetic mean* is the average of the input signal (block). It is computed from $v(n)$ or $p_v(v)$ by

$$\mu_v = \frac{1}{\mathcal{N}} \sum_{0}^{\mathcal{N}-1} v(n), \tag{3.12}$$

$$\mu_v = \sum_{v=-\infty}^{\infty} v \cdot p_v(v). \tag{3.13}$$

The result of Eq. (3.13) is also referred to as the *expected value*. The result of the arithmetic mean is a value between the minimum and maximum input signal value. The unit corresponds to the unit of the input signal. For symmetric PDFs, the arithmetic mean will be the x-axis position of the axis of symmetry. For example, a time domain audio signal usually has a mean value of approximately 0; if the signal has a DC offset, the mean will indicate the amount of the DC offset.

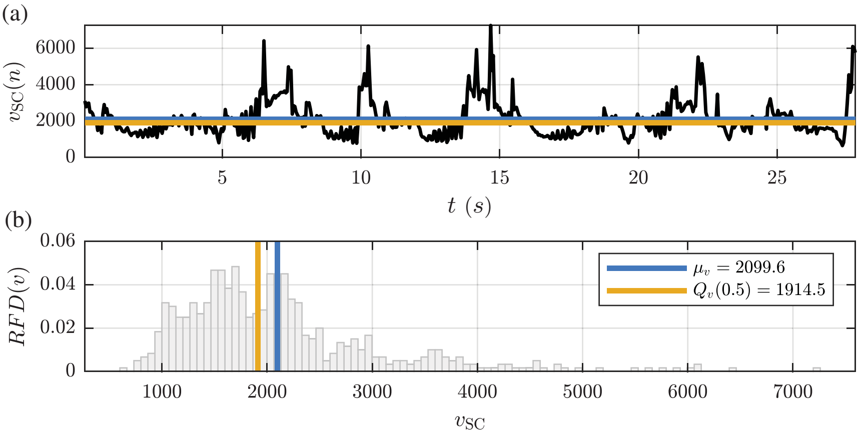
The calculation of an arithmetic mean is only of limited use if the PDF is not symmetric, see Figure 3.5. In this case, the computation of the median (see Section 3.1.3.5) of the PDF might be a more meaningful measure of the "average".

3.1.3.2 Geometric Mean

The *geometric mean* is an average measure for sets of positive numbers that are ordered on a logarithmic scale. It can be computed with

$$\mathrm{Mg}_v = \sqrt[\mathcal{N}]{\prod_{0}^{\mathcal{N}-1} v(n)} \tag{3.14}$$

$$= \exp\left(\frac{1}{\mathcal{N}} \sum_{0}^{\mathcal{N}-1} \log\left(v(n)\right)\right). \tag{3.15}$$



Repo 1: plotArithmeticMean.m

Figure 3.5 RFD (b) of a series of feature values (a) with its arithmetic mean μ_v and median $Q_v(0.5)$.

Equation (3.15) is equivalent to Eq. (3.14) but avoids problems with computational accuracy for long blocks of data and large values at the computational cost of applying a logarithm to each signal value. The result of the geometric mean is a value between the minimum and maximum input signal value. The unit corresponds to the unit of the input signal.

3.1.3.3 Harmonic Mean

The *harmonic mean* is a measure appropriate for averaging rates. It is

$$\mathrm{Mh}_v = \frac{\mathcal{N}}{\sum\limits_{0}^{\mathcal{N}-1} {}^{1}\!/\!{}_{v(n)}}. \tag{3.16}$$

3.1.3.4 Variance and Standard Deviation

Both the *variance* and the *standard deviation* measure the spread of the input signal $v(n)$ around its arithmetic mean. The variance σ_v^2 is defined by

$$\sigma_v^2 = \frac{1}{\mathcal{N}} \sum_{0}^{\mathcal{N}-1} \big(v(n) - \mu_v\big)^2, \tag{3.17}$$

$$\sigma_v^2 = \sum_{v=-\infty}^{\infty} (v - \mu_v)^2 \cdot p_v(v). \tag{3.18}$$

Strictly speaking, the former equation is the so-called *biased* estimate of the variance in contrast to the *unbiased* estimate

$$\sigma_{v,\mathrm{b}}^2 = \frac{1}{\mathcal{N} - 1} \sum_{0}^{\mathcal{N}-1} \big(v(n) - \mu_v\big)^2. \tag{3.19}$$

In case of $\mu_v = 0$, the variance equals the power of the observed block of samples.

The standard deviation σ_v can be computed directly from the variance

$$\sigma_v = \sqrt{\sigma_v^2}. \tag{3.20}$$

In case of $\mu_v = 0$, the standard deviation equals the Root Mean Square (RMS) of the observed block of samples (see Section 8.3.1). The result of the standard deviation is in the range

$$0 \leq \sigma_v \leq \max_{n \in [0;\mathcal{N}-1]} |v(n)|.$$

The standard deviation is 0 for silent or constant input signals. Its unit corresponds to the input signal's unit.

Figure 3.6 shows that, similar to the arithmetic mean, the standard deviation is only of limited use for nonsymmetric distributions.

3.1.3.5 Quantiles and Quantile Ranges

Quantiles can be computed from the PDF and can be used to divide the PDF into (equal-sized) subsets. They are helpful in the description of asymmetric distributions or distributions with so-called outliers, occasional untypical values far from the median (see below).

If the PDF is divided into two quantiles, it is split into two parts each containing 50% of the overall number of observations. The position of the border between those two quantiles will be referred to as the *median* $Q_v(0.5)$:

$$Q_v(0.5) = v\Bigg|_{\int\limits_{-\infty}^{v} p_v(y)\ dy = 0.5} \tag{3.21}$$

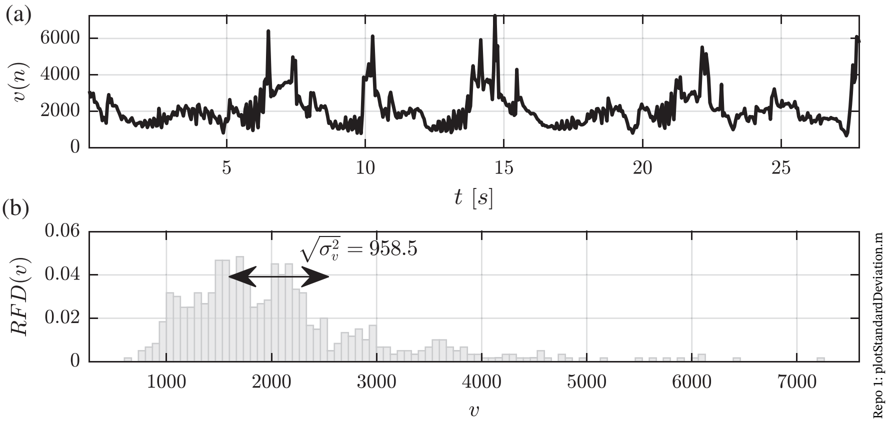


Figure 3.6 RFD (b) of a series of feature values (a) with the standard deviation.

which equals the arithmetic mean in the special case of symmetric distributions. The median is visualized next to the arithmetic mean in Figure 3.5.

Similarly, if the PDF is partitioned in four quantiles (so-called *quartiles*), the quantile borders would be $Q_v(0.25)$, $Q_v(0.5)$, and $Q_v(0.75)$. Figure 3.7 shows selected quantiles for a Gaussian and the skewed Chi-Squared distribution.

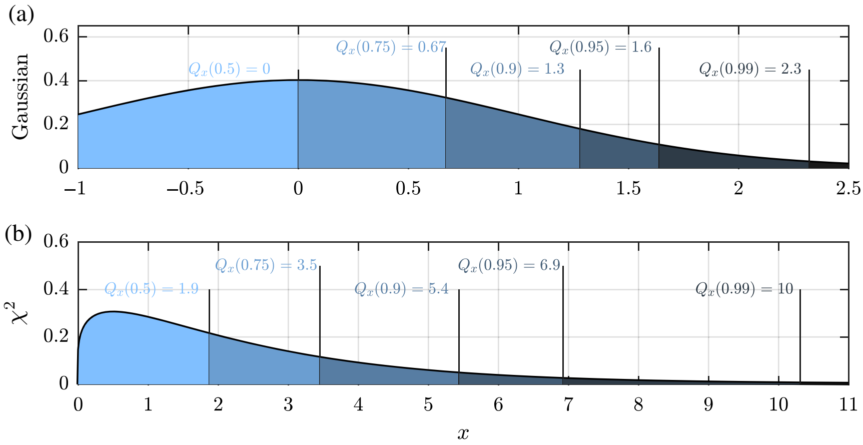
In many cases, *quantile ranges* are of specific interest for the simplified description of a distribution's shape. The range spanned by 90% of the samples can be computed by $\Delta Q_v(0.9) = Q_v(0.95) - Q_v(0.05)$. This is often seen as a good measure of the signal's range while discarding infrequent outliers, namely the upper and lower 5%.

The overall range of a signal is

$$\Delta Q_v(1.0) = \min\left(Q_v(1.0)\right) - \max\left(Q_v(0)\right). \tag{3.22}$$

3.1.4 Digital Audio Signals

Continuous real-world signals cannot be represented digitally. Instead, audio signals are represented as a series of values, so-called *samples*. This means it is required to discretize the input signal $x(t)$ into a series of (quantized) values at (equidistant) times $x(i)$. A quick introduction into this process is provided in Appendix A.1. Many of the properties of continuous signals also apply to discrete signals; the most important (and over-simplified) differences are that quantized (discretized in amplitude) signals will have a noise floor with a level depending on the quantizer resolution and sampled (discretized in time) signals have a



Repo 1: plotPdfQuantiles.m

Figure 3.7 Two probability distributions, Gaussian (a) and Chi-squared (b), with different quantiles marked.

limited bandwidth that is half of the sample rate. In theory and implementation, we often ignore that the signal is quantized and pretend that the amplitude is continuous. Typical audio sample rates range from 8 kHz for speech signals to 48 kHz or higher for music signals.

3.2 Audio Preprocessing

The raw audio signal is frequently preprocessed before computing instantaneous features (or more generally the input representation) from the data. The motivation for this preprocessing step is either to reduce the amount of data to be analyzed, to remove task-irrelevant information or to make the signal invariant to it, to remove redundant information, or to convert the signal into a format that is easier to process. Preprocessing, if applied correctly, can in many cases increase the robustness of the system.

Obviously, the options for preprocessing depend on the task; for instance, if we intend to determine the stereo width of a signal, we cannot down-mix it to mono. However, applying some preprocessing options even if they do not seem necessary, such as applying normalization before extracting a volume-independent feature such as the spectral centroid (see Section 3.5.1), may still be a good choice if the system should be invariant to this input property. This allows for better maintainability and easier extensibility of the system.

3.2.1 Down-Mixing

In many audio analysis problems, the information of interest can be represented by one single audio channel as well as by multiple input channels. For example, the tempo or information on the musical style should be extractable from a mono recording as well as from stereo or multichannel recordings.

Down-mixing is usually achieved by simply computing the arithmetic mean as introduced in Eq. (3.12) over all input channel signals $x_c(i)$ per sample i:

$$x(i) = \frac{1}{C}\sum_{c=0}^{C-1} x_c(i). \tag{3.23}$$

The number of input channels is C. It is also possible to apply different weights to different channels; the surround channels of a multichannel signal may, for example, have a lower weight than front channels. An alternative to down-mixing could be to use only one preselected audio channel; however, this may result in more loss of information if the channels have been mixed to produce a wide spatial image. Some audio applications also apply a phase shift of 90° to one channel before down-mixing a stereo signal to mono. The reason is to avoid a level boost of components present in all audio channels (mono components) as compared to single-channel components.

Listing 3.1 shows example code for downmixing an audio signal.

```
if (size(x, 2) > 1)
    x_downmix = mean(x, 2);
else
    x_downmix = x;
end
```

Repo 2: ToolDownmix.m
Repo 3: ToolDownmix.py

Listing 3.1 Matlab code snippet showing the down-mixing of a multichannel audio signal.

3.2.2 DC Removal

A DC offset – shown by a signal's arithmetic mean significantly different from zero – usually does not provide any useful information and may have unwanted impact on the following processing steps. Mathematically, the DC offset is simply removed by subtracting the arithmetic mean of the audio data $x_{\mathrm{DC}}(i)$ of length $\mathcal{I}$ (with $\mathcal{I}$ preferably representing the audio file length):

$$x(i) = x_{\mathrm{DC}}(i) - \frac{1}{\mathcal{I}}\sum_{i=0}^{\mathcal{I}-1} x_{\mathrm{DC}}(i). \tag{3.24}$$

In the case of real-time processing with a steady stream of subsequent samples, this can also be approximated by estimating the DC offset with a long Moving Average (MA) filter (see Appendix A.2.6.1) of a length $\mathcal{O}$

$$x(i) = x_{\mathrm{DC}}(i) - \frac{1}{\mathcal{O}}\sum_{j=i-\mathcal{O}/2}^{i+\mathcal{O}/2-1} x_{\mathrm{DC}}(j). \tag{3.25}$$

The number of coefficients has to be high to provide a reliable estimate of the arithmetic mean. Alternatively, any high-pass filter can be used to remove the DC part of the signal. The simplest filter is a differentiator:

$$x(i) = x_{\mathrm{DC}}(i) - x_{\mathrm{DC}}(i-1). \tag{3.26}$$

This differentiator, however, can have significant impact on higher-frequency components as well. This effect can be lessened by low-pass filtering the difference; the resulting DC removal filter would be

$$x(i) = (1 - \alpha) \cdot \big(x_{\mathrm{DC}}(i) - x_{\mathrm{DC}}(i-1)\big) + \alpha \cdot x(i-1) \tag{3.27}$$

with α between 0 and 1 being the coefficient of this single-pole filter.

3.2.3 Normalization

For the majority of analysis tasks, scaling the input amplitude should not affect the system output, as most musical content is not impacted by scaling the signal. In these cases, the signal can be *normalized* to have a predefined (maximum) amplitude or power.

A simple and commonly applied method to normalize an audio file is to detect the overall maximum of its absolute sample values and scale the signal so that this maximum's absolute value is mapped to 1:

$$x(i) = \frac{x_s(i)}{\max\limits_{\forall i}\left(|x_s(i)|\right)}. \tag{3.28}$$

Listing 3.2 shows example code for this audio signal normalization.

Repo 2: ToolNormalizeAudio.m
Repo 3: ToolNormalizeAudio.py

```
x_norm = x / max(abs(x), [], 'all');
```

Listing 3.2 Matlab code snippet showing the normalization of a multichannel audio signal.

Note that while this results in a normalized magnitude, it does not warrant equal loudness of different input files. Furthermore, the *normalization* may be influenced by signal distortions such as the clicks and crackles of a vinyl recording. In this case, the normalization to the maximum click amplitude will result in an "incorrect" scaling of the audio data.

The alternative is to use some other reference than the maximum for deriving the scaling factor, such as the RMS (see Section 8.3.1) or a loudness measurement with long integration time (see Section 8.1). In this case, additional processing may be necessary in order to avoid potential clipping of the scaled signal.

Normalizing a signal in a real-time context is difficult. Algorithms for automatic gain control or compressors and limiters provide a possible solution by monitoring the instantaneous input signal characteristics (e.g. the RMS or the peak level) and aiming at adjusting a time-variant gain value accordingly.

3.2.4 Sample Rate Conversion

Sample rate conversion can be a helpful processing step if (i) the sample rate of the input audio may vary between input files as, e.g. different music datasets may be recorded at different sample rates (e.g. 44.1, 48, 22.05 kHz), (ii) the system parametrization such as block length and hop size should be independent of audio sample rate, and/or (iii) reducing the sample rate helps to remove task-irrelevant high-frequency data and reduces the amount of audio samples to be processed.

In order to convert the sample rate from an input sample rate f_S to a target sample rate f_T, the number of samples per time unit has to be changed by a factor of l

$$f_\mathrm{T} = \frac{f_\mathrm{S}}{l}. \tag{3.29}$$

Taking into account the sampling theorem as given in Eq. (16.2) and the time scaling property of the spectrum as given in Eq. (17.9), it becomes clear that simply discarding or adding new samples might lead to aliasing artifacts. For example if $l > 1$ (*Down-sampling*), picking every lth sample can only work if the input signal $x(i)$ does not contain frequency components higher than ${}^{f_\mathrm{T}}/_{2}$; therefore, proper low-pass filtering is required when changing the sample rate.

Sample rate conversion with noninteger factors l is based on the same principles. The factor can be written as the ratio of two integer factors $l = {}^{l_1}/_{l_2}$ so that the signal can first be up-sampled by factor l_1 and then down-sampled by factor l_2. In between the two resampling steps, it is required to apply a low-pass filter which ensures both the reconstruction of the up-sampled signal and the suppression of aliasing artifacts in the down-sampled signal. There exist various approaches combining interpolation algorithms and low-pass filters for down-sampling a signal. One example for such a *band-limited* interpolation has been explained by Smith and Gosset and is usually referred to as *sinc* interpolation [3].

3.2.5 Block-Based Processing

Audio signal processing algorithms usually work block-based, meaning that the input signal is being split into consecutive blocks of samples with length $\mathcal{K}$. These blocks are processed one after another. In extreme cases, the block lengths can be either $\mathcal{K} = 1$ or equal to the length of the processed audio file, but in most cases, $\mathcal{K}$ is chosen arbitrarily between $\mathcal{K} = 32$ and $\mathcal{K} = 16,384$ (in many cases, powers of 2 are an advantage) depending on sample rate and required time resolution. Some reasons for this *block-based processing* are

- the internal usage of block-based algorithms such as the *Discrete Fourier Transform (DFT)* (see Section 3.3.1 and Appendix B),
- the characteristics of the used audio *Input/Output (IO)* hardware that usually returns blocks of samples,

- reduced memory allocation (instead of the complete audio file length; only the block length has to be allocated),
- enabling implementations capable of real-time processing, and
- increasing computational efficiency compared to individual sample processing through both vectorization with *Single Instruction Multiple Data (SIMD)* operations and reducing function call overhead.

Systems working in *real-time* have the constraint that the system has to be able to perform the processing of one block of data during the time frame this block consumes. Another condition of such systems is that they have no access to future input samples but only the present and possibly past samples. For real-time systems, the *latency* of the system, i.e. the delay between an input sample and the corresponding output value, can never be lower than the block length of the algorithm. Furthermore, the audio input block length does not necessarily correspond to the algorithm's internal block length. If the input block length is not a multiple of the internal block length, then efficiency problems can arise since the computational workload cannot be distributed uniformly over the input blocks, resulting in occasional workload peaks. Internally, most audio algorithms use overlapping blocks of data as depicted in Figure 3.8. That means that the boundaries of subsequently processed blocks overlap, i.e. that the last sample of the first block is after the first sample of the second block. An example of this blocking is shown in Figure 3.9. The sample indices of the block boundaries start sample $i_s(n)$ and stop sample $i_e(n)$ of processing block n can then be formulated as follows:

$$i_s(n) = i_s(n-1) + \mathcal{H}, \tag{3.30}$$

$$i_e(n) = i_s(n) + \mathcal{K} - 1, \tag{3.31}$$

with $\mathcal{H}$ being the so-called *hop size* in samples. The hop size should be smaller than or equal to the block length. The *block overlap ratio* can be calculated with

$$o_r = \frac{\mathcal{K} - \mathcal{H}}{\mathcal{K}}. \tag{3.32}$$

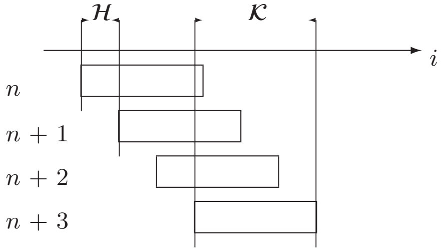
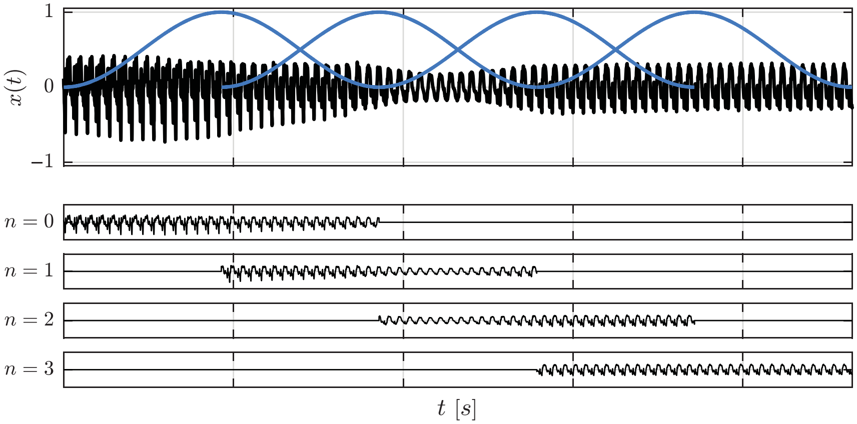


Figure 3.8 Schematic visualization of block-based processing: the input signal with the input sample index i is split into overlapping blocks of length $\mathcal{K}$ and block index n; the hop size $\mathcal{H}$ is the distance between the start of one block and the following block.



Repo 1: plotBlocking.m

Figure 3.9 Example visualization of blocking: the input signal (top) is split into four overlapping blocks.

A typical block overlap ratio is $o_\mathrm{r} \geq 1/2$. If one *time stamp* is assigned to a processing block, the assigned time stamp in samples $t_\mathrm{s}(n)$ can usually be either the start frame of the block $t_\mathrm{s}(n) = i_\mathrm{s}(n)/f_\mathrm{S}$ or its middle position:

$$t_\mathrm{s}(n) = \frac{i_\mathrm{e}(n) - i_\mathrm{s}(n) + 1}{2 \cdot f_\mathrm{S}} + \frac{i_\mathrm{s}(n)}{f_\mathrm{S}} = \frac{\mathcal{K}}{2 \cdot f_\mathrm{S}} + \frac{i_\mathrm{s}(n)}{f_\mathrm{S}}. \tag{3.33}$$

If the time stamp $t_\mathrm{s}(n)$ is chosen to be in the middle of the block, the practical problem arises that the start and end time stamps always represent a shorter time range than the audio signal itself. For example, if the audio signal starts at sample index 0, the first time stamp will be at sample index $(\mathcal{K}-1)/2$. This can be avoided by padding the signal with $\mathcal{K}/2$ zeros at the beginning (and the end). Note that different blocking implementations might choose to pad a different amount of zeros to begin (typically ranging from 0 to $(\mathcal{K} - \mathcal{H})$) and end (typically ranging from 0 to $\mathcal{K}$) of the signal. This might lead to differences between implementations in the number of blocks even in the case of identical block lengths and hop sizes. Larger amounts of zeros ensure that no audio samples are discarded regardless of the input audio length; however, the results for the first and the last block may be unreliable due to a potentially large number of zeros. After the padding, the blocking itself is a simple implementation as shown in Listing 3.3.

Each individual audio block can be interpreted as the complete signal multiplied with a window function (see also Appendix B.4).

The individual audio blocks are then subjected to further processing, e.g. computing a DFT from the block of audio data (see Section 3.3.1) or extract a feature for this block (see Section 3.5). These results per block might also be aggregated in so-called texture windows (see Section 3.7.2).

```
% pad with zeros just to make sure it runs for weird inputs, too
xPadded = [x; zeros(iBlockLength+iHopLength, 1)];

x_b = zeros(iNumBlocks, iBlockLength);

for n = 1:iNumBlocks
    x_b(n,:) = xPadded((n-1)*iHopLength+1:(n-1)*iHopLength+iBlockLength);
end
```

Repo 2: ToolBlockAudio.m
Repo 3: ToolBlockAudio.py

Listing 3.3 Matlab code snippet for blocking an audio file into overlapping blocks.

3.2.6 Other Preprocessing Options

As with the selection of appropriate input representations, the preprocessing options have to always be selected with the application in mind. Every preprocessing step which improves the algorithm's accuracy or its robustness, minimizes its complexity or computational workload, or simplifies handling the data is beneficial. In addition to the presented preprocessing options, it is, for example also common to attenuate the level of any unwanted frequency region by applying a filter to the signal.

3.3 Time-Frequency Representations

The vast majority of advanced audio analysis and processing systems do not utilize the time domain series of samples directly; instead, blocks of the signal are commonly transformed to the frequency domain for subsequent processing. The reasons for this are (i) a more intuitively accessible representation of the audio data that directly conveys frequency components, and (ii) the often more sparse representation of the audio data in the spectral domain.

3.3.1 Fourier Transform

The *Discrete Fourier Transform (DFT)* is one of the most important tools for processing and analyzing audio signals. A more detailed discussion of its properties – starting with the Fourier Transform (FT) of continuous signals – can be found in Appendix B. The DFT of a block of the signal $x(i)$ is defined by

$$\mathfrak{F}\left\{x_{\text{block}}(i)\right\} = \sum_{i=0}^{\mathcal{K}-1} x_{\text{block}}(i) e^{-\mathrm{j}ki\Delta\Omega} \tag{3.34}$$

with $\Delta\Omega$ being the distance between two frequency bins as angular frequency difference

$$\Delta\Omega = \frac{2\pi}{\mathcal{K}}. \tag{3.35}$$

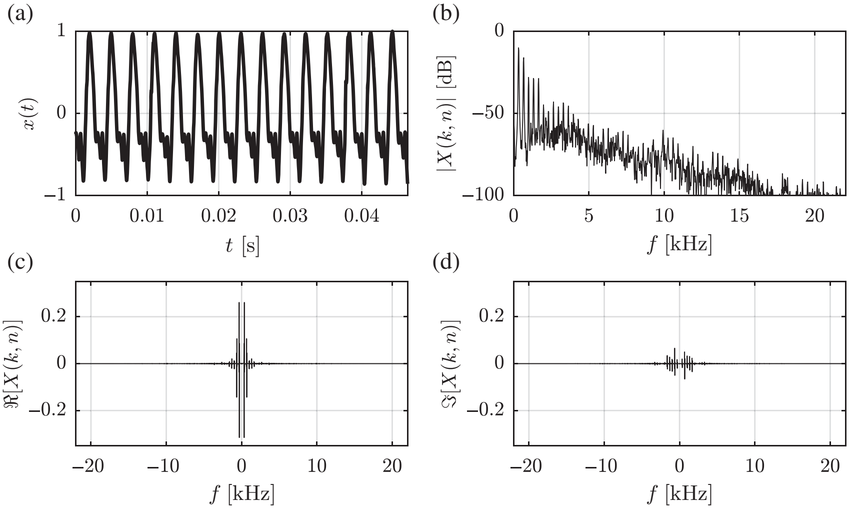
The frequency of bin index k can then be computed from the block length $\mathcal{K}$ and sample rate f_S by

$$f(k) = \frac{\Delta\Omega}{2\pi} k f_\mathrm{S} = \frac{k}{\mathcal{K}} f_\mathrm{S}. \tag{3.36}$$

We will refer to the DFT of the nth block with length $\mathcal{K} = i_\mathrm{e}(n) - i_\mathrm{s}(n) + 1$ of the audio data as *Short Time Fourier Transform (STFT)*:

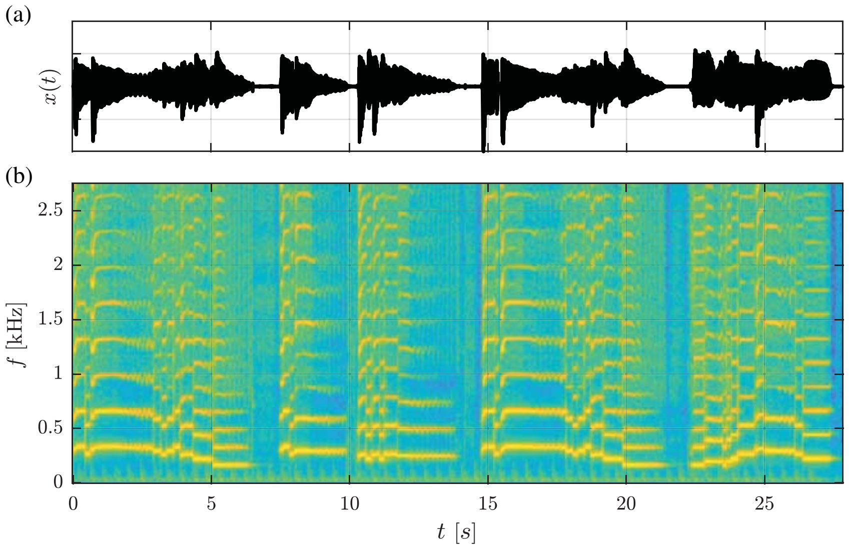
$$X(k,n) = \sum_{i=i_\mathrm{s}(n)}^{i_\mathrm{e}(n)} x(i) \exp\left(-\mathrm{j}k \cdot \left(i - i_\mathrm{s}(n)\right) \frac{2\pi}{\mathcal{K}}\right). \tag{3.37}$$

Figure 3.10 shows a block of a real-world audio signal, its magnitude spectrum, as well as its real and imaginary spectrum. Typical block lengths for audio range from 256 to 16 384 depending on the sample rate and the requirements on time and frequency resolution. A fast implementation of the DFT is the *Fast Fourier Transform (FFT)*.



Repo 1: plotFourierTransform.m

Figure 3.10 Short-Time Fourier Transform: time domain block (a), magnitude spectrum (b), real spectrum (c), and imaginary spectrum (d).



Repo 1: plotSpecgram.m

Figure 3.11 Time-domain waveform (a) and spectrogram visualization (b); each column represents (the magnitude of) a STFT of a block with the time increasing from left to right and the level of each specific point in frequency and time indicated by the color or brightness of each bin.

Understanding the properties of the FT is a helpful prerequisite for the design of many frequency-domain algorithms. See Appendix B for a list of important properties and their derivations.

A typical visualization of an audio signal that combines time and frequency aspects is the *spectrogram* as shown in Figure 3.11. An STFT is calculated for each (overlapping) block of sample data; the resulting magnitude spectra are then plotted in a pseudo-three-dimensional image. Each column is the magnitude spectrum of one block, and each bin is darkened or colored according to its level. The input of the majority of ACA systems is either the waveform as discussed above or, more commonly, some kind of spectrogram representation. The spectrogram representation often provides more insights than the time-domain waveform as it visualizes the frequency content as it changes over time.

Figure 3.11 shows the first 24 bars of a single saxophone playing the jazz standard *Summertime*; while the time domain waveform allows us to identify melodic phrases through observation of general amplitude fluctuations, the spectrogram also shows that (i) it is a recording of a single monophonic instrument (clear spectral structure of fundamental frequency and harmonics at integer multiples), (ii) it is an instrument that allows vibrato (see seconds 3 and 18), (iii) it is an instrument with a significant number of harmonics (number and strength of "parallel" lines), and that (iv) the melody can be directly derived from the visualization by identifying the lowest frequency of the harmonic series and mapping it to musical pitch.

Listing 3.4 shows the computation of a spectrogram from the blocked audio signal

```
for n = 1:size(X,2)
    tmp = fft(x_b(n, :)' .* afWindow);

    if bMagnitude
        X(:, n) = abs(tmp(1:size(X, 1))) * 2 / iBlockLength;
    else
        X(:, n) = (tmp(1:size(X, 1))) * 2 / iBlockLength;
    end
end

% normalization
X([1 end],:) = X([1 end],:) / sqrt(2);
```

Repo 2: computeSpectrogram.m
Repo 3: computeSpectrogram.py

Listing 3.4 Matlab code snippet showing the over blocks for spectrogram computation.

3.3.2 Constant *Q* Transform

The linear frequency axis of the STFT does not match the human perception of pitch well. Two neighboring pitches with a specific musical pitch distance will have a smaller frequency distance in Hertz if they are low-frequency pitches than if they are high-frequency pitches (see Section 7.1). The *Constant Q Transform (CQT)* addresses this issue by adapting the STFT lengths per frequency and calculates frequency coefficients similar to the DFT but on a logarithmic frequency scale [4]:

$$X(k,n) = \frac{1}{\mathcal{K}(k)} \sum_{i=i_\mathrm{s}(n)}^{i_\mathrm{e}(n)} w_k(i - i_\mathrm{s}) \cdot x(i) \exp\left(\mathrm{j}2\pi \frac{Q \cdot (i - i_\mathrm{s})}{\mathcal{K}(k)}\right). \tag{3.38}$$

Q adjusts the desired frequency resolution and can be derived from c, the desired number of frequency bins per octave, while $\mathcal{K}(k)$ depends on both the target frequency and the *quality factor* Q:

$$Q = \frac{f}{\Delta f} = \frac{1}{2^{1/c} - 1}, \tag{3.39}$$

$$\mathcal{K}(k) = \frac{f_\mathrm{S}}{f(k)} Q. \tag{3.40}$$

The CQT can be interpreted as computing a DFT only for specific, logarithmically spaced, frequency bins. Figure 3.12 schematically visualizes two ways of implementing the CQT. On the left, the largest window length is split into smaller lengths so that many STFTs of different lengths are computed. On the right, all STFTs are of the same size, but the input content is zero-padded. The latter approach requires a hop length of the shortest block length, as opposed to the first approach, where the hop length equals the longest block length.

The CQT is not invertible and can in terms of efficiency be compared to a DFT (as opposed to the *FFT*). There are approaches to make the CQT both computationally more efficient and quasiinvertible, see, e.g. [5, 6].

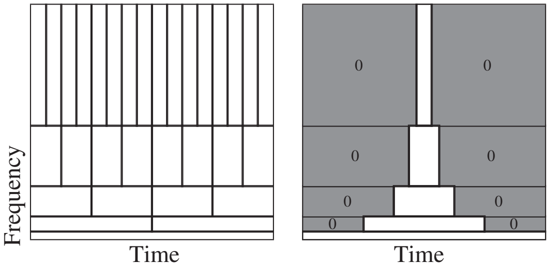


Figure 3.12 Two approaches of implementing blocking for the CQT: compute multiple STFTs with different block lengths (left) and compute multiple zero-padded STFTs of equal block length.

3.3.3 Log-Mel Spectrogram

The *Log-Mel Spectrogram* is nowadays probably the most commonly used input representation for Deep Neural Networks (DNNs) as it seems to be a fitting representation for transporting condensed timbre and pitch information. It is computed from the magnitude spectrogram by grouping STFT bins together in overlapping frequency bands that approximate the pitch perception of the human ear as modeled by the Mel scale (see Section 7.1.1). Therefore, the frequency resolution will (roughly) logarithmically decrease with increasing frequency. This can be easily seen by comparing the distance between low and high harmonics in Figures 3.11 and 3.13. The number of frequency bands of the Mel Spectrogram is usually significantly lower than the number of frequency bins of the STFT. This Mel Spectrogram becomes the Log-Mel Spectrogram if a logarithm is applied to the spectral magnitudes in order to approximate human loudness perception (see Section 8.1).

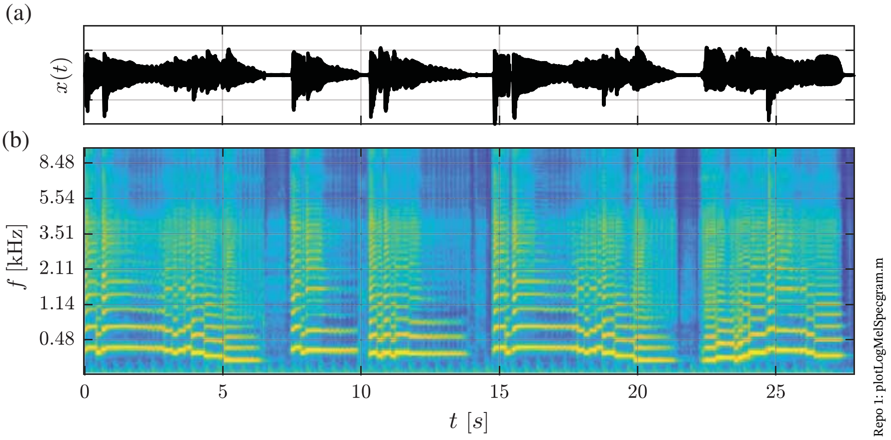


Figure 3.13 Time-domain waveform (a) and corresponding Log-Mel Spectrogram (b) of a monophonic saxophone signal; similarly to a normal spectrogram, time increases linearly from left to right with the hop length, but the frequency axis is now nonlinear (harmonics are not equidistant).

Figure 3.13 visualizes the Log-Mel Spectrogram of the audio sample shown in Figure 3.11.

Similar to the CQT, the Log-Mel Spectrogram has a nonlinear frequency axis and is not invertible. Its computation, however, tends to be similarly efficient as the STFT as it only applies a few simple operations on the STFT (see Listing 3.5).

```
[X, f, t] = ComputeSpectrogram(x, ...
                               f_s, ...
                               afWindow, ...
                               iBlockLength, ...
                               iHopLength);

% compute mel filters
[H, f_c] = getMelFb_I(iBlockLength, f_s, iNumMelBands, fMax);

% apply mel filters
M = H*X;

% amplitude to level
if (bLogarithmic)
    M = 20*log10(M+1e-12);
end
```

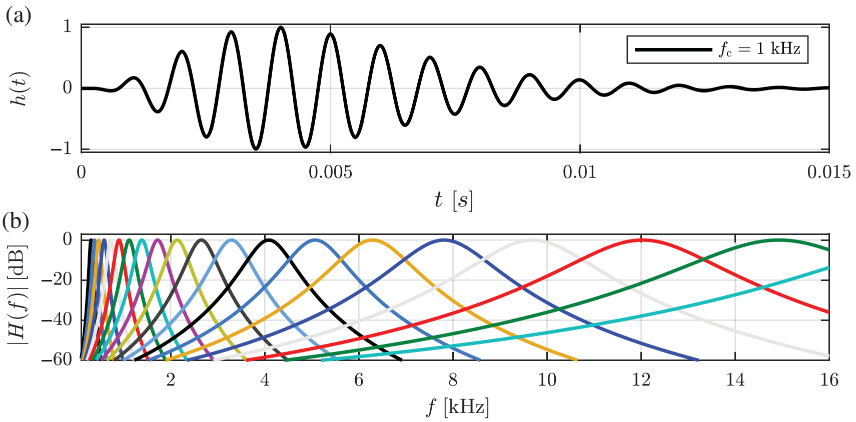
Repo 2: computeMelSpectrogram.m
Repo 3: computeMelSpectrogram.py

Listing 3.5 Computation of the Mel Spectrogram by grouping bins of the magnitude spectrum.

3.3.4 Filterbanks

An *auditory filterbank* is frequently used in psycho-acoustical or physiological ear models. It approximates the resolution and selectivity of the human ear. The filterbank's midfrequencies are usually computed according to the Mel scale (see Section 7.1).

In signal processing applications, the usage of auditory filterbanks is not as widespread as one might assume. The most obvious reasons are (i) the computational workload produced by a filterbank with reasonable frequency resolution and (ii) the restriction to analysis tasks since the computed frequency domain representation cannot be transformed back to the time domain (except for a few carefully designed filterbanks with other limitations). But in the case of ACA, it has not been consistently shown that approaches using a filterbank give more reliable results than, e.g. an STFT-based analysis. One possible explanation is that in many cases, the subject of investigation is not what humans are able to hear in a signal but physical properties of the signal. Still, the argument that ultimately every audio analysis task is about the extraction of *perceivable* features has merit as well, and the frequent use of the Mel Spectrogram with its STFT filters emphasizes the usefulness of the logarithmic frequency axis. Auditory modeling is a lively subject of research; Lyon et al. review different approaches in [7].



Repo 1: plotGammatone.m

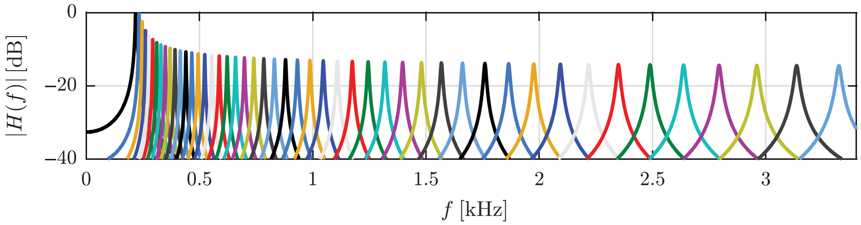
Figure 3.14 Normalized impulse response of a gammatone filter with a center frequency $f_c = 1$ kHz, a bandwidth $\Delta f = 125$ Hz, and an order $\mathcal{O} = 4$ (a) and magnitude frequency response of a gammatone filterbank with 20 bands between 100 Hz and 24 kHz (b).

A widely used auditory filterbank is the so-called *gammatone filterbank*. A gammatone filter has an impulse response of the form [8, 9]:

$$h(i) = \frac{A\cdot \left({}^{i}\!/_{f_S}\right)^{\mathcal{O}-1}\cdot \cos\left(2\pi\cdot f_c\frac{i}{f_S}\right)}{e^{2\pi i\Delta f/f_S}}. \tag{3.41}$$

Figure 3.14 shows the impulse response of one gammatone filter as well as the frequency (magnitude) response of a gammatone filterbank for 20 bands between 100 Hz and 24 kHz. Slaney showed that gammatone filters can be efficiently implemented with cascaded second-order filters [10]. A survey of different gammatone filter implementations can be found in [11].

Nonauditory filterbanks have also been proposed for audio analysis. For instance, Lerch used a bank of resonance filters with one filter for each semitone as shown in Figure 3.15 [12].



Repo 1: plotResonanceFilterbank.m

Figure 3.15 Frequency response of a resonance filterbank spanning four octaves with one filter per semitone starting at the pitch A3.

3.4 Other Input Representations

Many other input representations for audio analysis systems have been proposed and used. These include Wavelet Transforms [13], Correlograms [14], and various other representations. It is hard to systematically categorize them; sometimes, the input representation of one system might be the midlevel representation of another system. Two task-specific input representations, the beat histogram and the self similarity matrix will be introduced later in Sections 9.4 and 9.7, respectively.

3.5 Instantaneous Features

Historically, the input representation of audio analysis systems commonly used to be a set of *features*. These features are usually a low-dimensional, task-relevant representation of the audio signal. Expert knowledge is applied to selecting and designing such task-relevant features to allow the inference system to focus on what experts deem the meaningful properties of the audio signal. The more powerful the feature representation is, the less powerful the classifier or other inference algorithm needs to be (and vice versa). The potential disadvantage of using this reduced feature representation is that the features might not represent *all* relevant information of the audio signal; however, once stripped away, information not represented by the features is unavailable to the inference system.

Although many state-of-the-art systems have moved away from audio feature input and are utilizing more generic input representations such as the Log-Mel Spectrogram, feature-based approaches are still useful for tasks with limited amounts of annotated training data for which data-hungry neural networks might overfit too fast. In these cases, a feature-based approach can be preferable as it reduces the required complexity of the machine-learning algorithm by utilizing expert knowledge in selecting the features. Not only because of this reason, but also because it makes sense didactically, this section introduces a selection of widely used features that have been used extensively for audio content analysis. It is important to note, however, that the choice of the specific features used in an algorithm will ultimately be driven by the task to solve; this can make the modification of well-known features as well as the design of new features advantageous or even mandatory. Therefore, the number of possible features used in audio content analysis is probably limitless and only a somewhat representative set can be presented in the following.

The term *instantaneous feature*, *short-term feature*, *low-level feature*, or *audio descriptor* is generally used for measures that generate one value per (short) block of audio samples. An instantaneous feature is not necessarily musically, musicologically, or perceptually meaningful all by itself, and is therefore frequently referred to as a *low-level feature*. A low-level feature can serve as a building block for the construction of higher-level features describing more meaningful properties of the (music) signal. While there is general consensus about the term low-level features (usually extracted from short blocks of audio), there is less agreement on what characterizes a mid-level or high-level feature beyond the general notion that with increasing level the musical, semantic, and/or perceptual meaning increases. Without trying to formally define something that is probably not meaningful to define, examples for mid-level and high-level features could be tempo and musical genre, respectively.

The following properties are desirable for a feature to have:

- *high "discriminative" or descriptive power* since the feature should be suitable and relevant to the task at hand,
- *noncorrelation to other features* because each feature should add new information to avoid redundancy,
- *invariance to irrelevancies* to allow the feature to be robust against, e.g. linear transformations of the input audio signal such as scaling and filtering operations (low-pass filtering, reverberation), the addition of signals such as (background) noise, coding artifacts, as well as the application of nonlinear operations such as distortion and clipping (see Wegener et al. for an example evaluation of feature robustness [15]), and
- *reasonable computational complexity* to ensure that the feature is able to be computed on the target platform (such as a mobile device) and for the required application, respectively.

Note that this list is neither exhaustive nor are all items necessarily required, but it is a good starting point when compiling a feature set.

A simple example for an instantaneous feature is the magnitude or amplitude level of the audio signal extracted on a block-per-block basis. It represents a widely known reduced representation of the audio signal, the waveform. Although simple to extract, it may show characteristics usable for the extraction of higher-level information. This is demonstrated by Figure 3.16 which illustrates three waveforms of length 15 s, a speech signal, a chamber music signal, and a pop music signal. The general shape of the waveform envelope already enables an observer to differentiate between these different signals by deriving descriptive properties related to dynamic range, dynamic fluctuations, and pauses. This is, of course, not necessarily true for every possible excerpt from these genres, but it tells us that something as simple as an envelope description may be useful in the automatic detection of a signal type or musical genre.

Several attempts have been made to categorize instantaneous features into different taxonomies, compare the MPEG-7 standard [16] or the categories proposed

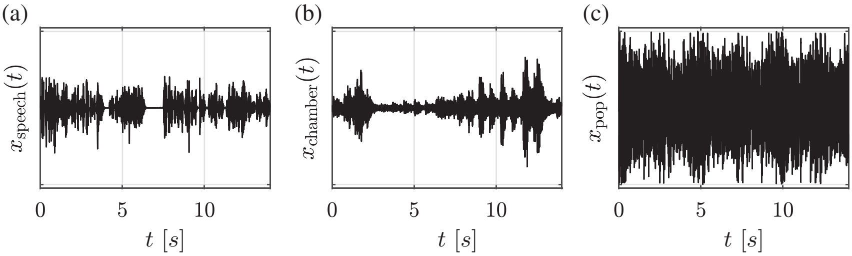


Figure 3.16 Waveform of excerpts from a speech recording (a), a string quartet recording (b), and a pop recording (c) with a length of 15 seconds, respectively.

by Peeters [17]. It turns out that it is difficult to find a simple and consistent yet practically useful feature categorization that is task-agnostic, provides a useful level of detail, and ensures that categories are not overlapping. Therefore, we abandon elaborate categorization attempts and simply list a variety of commonly used features in the following:

Figure 3.17 visualizes the process of extracting a feature value $v(n)$ from one block with index n with the block length

$$\mathcal{K} = i_{\mathrm{e}}(n) - i_{\mathrm{s}}(n) + 1. \tag{3.42}$$

Most features extracted from a frequency domain representation of a signal describe the spectral shape of the audio signal and are closely related to the

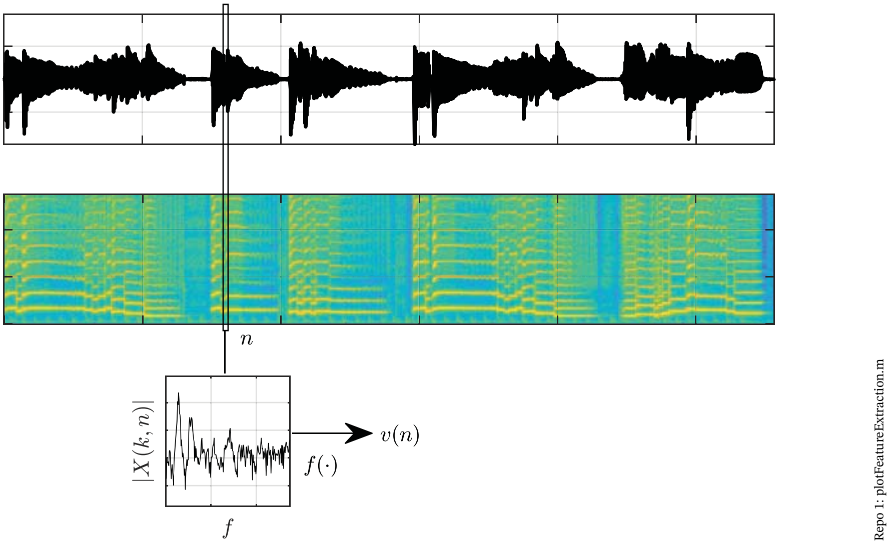


Figure 3.17 Visualization of the feature extraction process.

timbre of this signal. The *timbre* of a sound is referred to as its *sound color*, its *quality*, or its *texture*. Besides pitch and loudness, timbre is considered as “the third attribute of the subjective experience of musical tones” [18]. Loudness and pitch are unidimensional properties, as sounds with different loudness or pitch can be ordered on a single scale from quiet to loud and low to high, respectively. Timbre, however, is a multidimensional property [19, 20], which complicates its definition. The most infamous example is probably the definition of the American Standards Association from 1960 that defined timbre as “that attribute of auditory sensation in terms of which a listener can judge that two sounds similarly presented and having the same loudness and pitch are dissimilar” [21]. This definition has been criticized repeatedly by researchers mainly because, as Bregman points out [22], it (i) does not attempt to explain what timbre is, but only what timbre is *not*, i.e. loudness and pitch, and (ii) implies that timbre only exists for sounds with a pitch, implicating that, for example percussive instruments do not have a timbre. Early use of the term *timbre* can be found in Blumenbach [23]. Helmholtz was probably the first to detect the dependency between the timbre of a sound and the relative amplitudes of the harmonics during the second half of the nineteenth century [24]. Although he noted that other influences play a role in defining the quality of a tone such as the “beginning” and “ending” of a sound, he restricted his definition of timbre (“Klangfarbe”) to the harmonic amplitude distribution only. Stumpf argued that timbre is influenced by more attributes [25]. He named not only the relative amplitude of harmonics but also the form and length of the attack time and note endings, and additional sounds and noise as the third timbre-determining component. Seashore proposed to restrict the term timbre during the first half of the twentieth century to the harmonic structure that “is expressed in terms of the number, distribution, and relative intensity of its partials,” and additionally proposed the term *sonance*, referring to “the successive changes and fusions which take place within a tone from moment to moment” [26]. This distinction, however, did not find broad acceptance in the research community. Nowadays, timbre is understood as the phenomenon that takes into account both spectral patterns and temporal patterns [20, 27]. Timbre perception is obviously influenced by numerous parameters of both the onset properties such as rise time and inharmonicities during the onset as well as numerous steady-state effects such as vibrato, tremolo, and pitch instability [18].

In the following, we will restrict ourselves to measures of spectral shape since most of the features describing “temporal timbre” only work for individual monophonic notes as opposed to complex time-variant mixtures of signals. Many of the presented spectral feature definitions are technically motivated; therefore, even if they describe the spectral shape of a signal, there might be no direct relation to any dimension of human timbre perception.

3.5.1 Spectral Centroid

The *spectral centroid* represents the Center of Gravity (COG) of spectral magnitudes. In other words, the magnitude spectrum is interpreted as a distribution and the spectral centroid is the mean of this distribution (compare Eq. (3.13)). Since the magnitude spectrum is – unlike the typical distribution – not normalized to a sum of 1, the spectral centroid is the frequency-weighted sum of the magnitude spectrum normalized by its unweighted sum:

$$v_{\mathrm{SC}}(n) = \frac{\sum\limits_{k=0}^{\mathcal{K}/2} k \cdot |X(k,n)|}{\sum\limits_{k=0}^{\mathcal{K}/2} |X(k,n)|}. \tag{3.43}$$

The result of the spectral centroid is a bin index within the range $0 \leq v_{\mathrm{SC}}(n) \leq \mathcal{K}/2$.[1] It can be converted either to Hertz by using Eq. (3.36) or to a parameter range between zero and one by dividing it by the maximum frequency index $\mathcal{K}/2$. Low results indicate significant low-frequency components and insignificant high-frequency components. As the spectral centroid is undefined for silent input frames, special consideration is required for frames containing only zeros (see Listing 3.6).

```
vsc = ((0:size(X, 1)-1)*X) ./ sum(X, 1);

% avoid NaN for silence frames
vsc (sum(X, 1) == 0) = 0;

% convert from index to Hz
vsc = vsc / (size(X, 1)-1) * f_s / 2;
```

Repo 2: FeatureSpectralCentroid.m
Repo 3: FeatureSpectralCentroid.py

Listing 3.6 Implementation of the *spectral centroid* computed from $|X(k,n)|$.

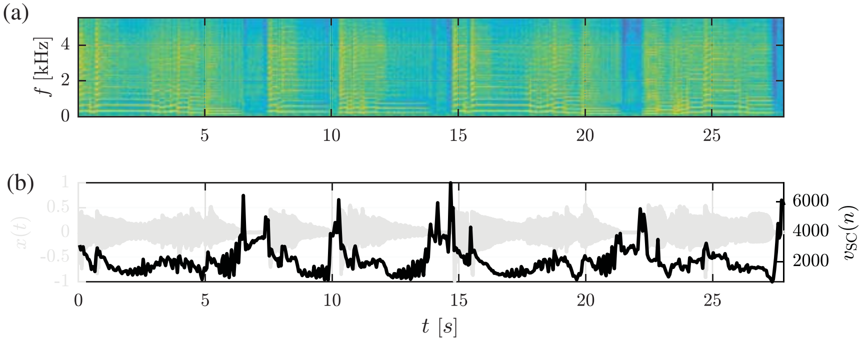
Figure 3.18 shows the spectral centroid for the example saxophone signal introduced above. In the case of this monophonic signal, one can see how the spectral centroid moves with the fundamental frequency during tonal parts, spikes at initial transients, and is high during pauses because of the noise in the audio signal.

Numerous indications can be found in the literature that the spectral centroid is highly correlated with the timbre dimension *brightness* or *sharpness* [28–34].

Note that the Listing 3.6 actually returns the spectral centroid in Hertz, not in bins as given in Eq. 3.43. Differences such as these are very common between

---

[1] Note that all spectral domain equations in this text cover all $\mathcal{K}/2 + 1$ values of the magnitude spectrum. Practically, the frequency bin with index $\mathcal{K}/2$ can be omitted as it does not provide reliable information.



Repo 1: plotFeature.m

Figure 3.18 Spectrogram (a), waveform (b, background), and *spectral centroid* (b, foreground) of a saxophone signal.

different implementations.[2] It is also common to compute the spectral centroid from the power spectrum instead of the magnitude spectrum:

$$v_{\mathrm{SC}}(n) = \frac{\sum\limits_{k=0}^{\mathcal{K}/2} k \cdot |X(k,n)|^2}{\sum\limits_{k=0}^{\mathcal{K}/2} |X(k,n)|^2}. \tag{3.44}$$

Zwicker and Fastl presented a psychoacoustic model of sharpness that uses the excitation patterns to compute the sharpness [35]. It differs from Eq. (3.43) mainly in two points. First, it is computed on a nonlinear Bark scale, the so-called critical band rate which models the nonlinearity of human frequency perception (see Section 7.1.1). Second, it utilizes a psychoacoustic loudness measure instead of the spectral power; this loudness model takes into account masking and other perceptual effects. While ignoring the difference between loudness and power, the idea of a perceptually motivated nonlinear frequency scale has been adapted for the definition of the spectral centroid in the MPEG-7 standard [16]. The critical band rate is approximated by applying a logarithm to the frequencies with a reference point of $f_{\mathrm{ref}} = 1000$ Hz:

$$v_{\mathrm{SC,log}}(n) = \frac{\sum\limits_{k=k(f_{\min})}^{\mathcal{K}/2} \log_2\left(\frac{f(k)}{f_{\mathrm{ref}}}\right) \cdot |X(k,n)|^2}{\sum\limits_{k=k(f_{\min})}^{\mathcal{K}/2} |X(k,n)|^2}. \tag{3.45}$$

In this specific MPEG-definition, all bins corresponding to frequencies below 62.5 Hz are combined to one band with a midfrequency of 31.25 Hz.

---

[2] Given that most features will be normalized across the training data, differences in feature scaling do not usually change the system results.

The unnormalized spectral centroid, i.e. the numerator of Eq. (3.43), is sometimes referred to as *High Frequency Content (HFC)* [36].

3.5.2 Spectral Spread

The *spectral spread*, sometimes also referred to as *instantaneous bandwidth* [37], describes the concentration of the magnitude spectrum around the spectral centroid. As the spectrum is interpreted as a distribution, the spectral spread can be seen as the standard deviation of this distribution (compare Eq. (3.20)) around its spectral centroid. Its definition is

$$v_{\mathrm{SS}}(n) = \sqrt{\frac{\sum\limits_{k=0}^{\mathcal{K}/2} \left(k - v_{\mathrm{SC}}(n)\right)^2 \cdot |X(k,n)|}{\sum\limits_{k=0}^{\mathcal{K}/2} |X(k,n)|}}. \qquad (3.46)$$

The result of the spectral spread is a bin range of $0 \leq v_{\mathrm{SS}}(n) \leq {}^{\mathcal{K}}\!/_{4}$. It can be converted either to Hertz by using Eq. (3.36) or to a parameter range between zero and one by dividing it by ${}^{\mathcal{K}}\!/_{4}$. Low results indicate the concentration of the spectral energy at a specific frequency region and high values indicate a broad, noise-like spectrum. Similar to the spectral centroid, the spectral spread is not defined for audio blocks with no spectral energy (see Listing 3.7).

```
% get spectral centroid as index
vsc = FeatureSpectralCentroid(X, f_s) * 2 / f_s * (size(X, 1)-1);

% allocate memory
vss = zeros(size(vsc));

% compute spread
for n = 1:size(X,2)
    vss(n) = (((0:size(X, 1)-1)-vsc(n)).^2 * X(:, n)) ./ sum(X(:, n));
end
vss = sqrt(vss);

% convert from index to Hz
vss = vss / (size(X, 1)-1) * f_s / 2;

% avoid NaN for silence frames
vss (sum(X, 1) == 0) = 0;
```

Repo 2: FeatureSpectralSpread.m
Repo 3: FeatureSpectralSpread.py

Listing 3.7 Implementation of the *spectral spread* computed from $|X(k,n)|$.

Figure 3.19 shows the spectral spread for an example signal. Most prominent are the high-feature values during pauses and at transients; the spectral spread is generally low during steady pitches for this monophonic signal.

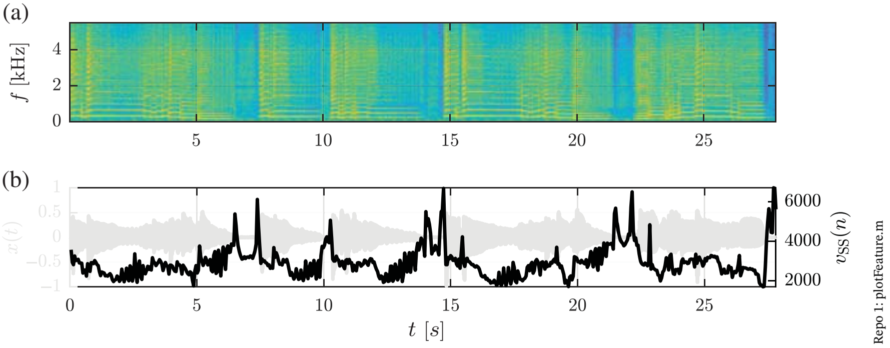


Figure 3.19 Spectrogram (a), waveform (b, background), and *spectral spread* (b, foreground) of a saxophone signal.

There are indications that the spectral spread is of some relevance in describing the perceptual dimensions of timbre [32].

The calculation of the spectral spread has to conform with the definition of the spectral centroid. For example, if the spectral centroid has been calculated from the power spectrum instead of the magnitude spectrum, then the spectral spread needs to be calculated from the power spectrum as well. If the spectral centroid follows the MPEG-7 definition mentioned above, a logarithmic frequency scale has to be used for the calculation of the spectral spread as well:

$$v_{\mathrm{SS,log}}(n) = \sqrt{\frac{\sum\limits_{k=k(f_{\min})}^{\mathcal{K}/2} \left(\log_2\left(\frac{f(k)}{f_{\mathrm{ref}}}\right) - v_{\mathrm{SC}}(n)\right)^2 \cdot |X(k,n)|^2}{\sum\limits_{k=k(f_{\min})}^{\mathcal{K}/2} |X(k,n)|^2}}. \tag{3.47}$$

3.5.3 Spectral Skewness and Spectral Kurtosis

The *skewness* and *kurtosis* are derived from the third- and fourth-order central moments of a distribution, respectively (see also the variance in Section 3.1.3.4 as second-order moment). The central moment of order $\mathcal{O}$ is defined by

$$\gamma_{v,\beta} = \sum_{0}^{\mathcal{N}-1} \left(v(n) - \mu_v\right)^{\mathcal{O}}. \tag{3.48}$$

The skewness is useful as a measure of the asymmetry of a PDF. It will be 0 for symmetric distributions, negative for distributions with their mass centered on the right (left-skewed), and positive for distributions with their mass centered on the left (right-skewed). Note that while every symmetric distribution has zero skewness, the converse is not necessarily true. The kurtosis is a measure of "non-Gaussianity" of a PDF; more specifically, it indicates the flatness (and peakiness, respectively) of the input values' distribution compared to the Gaussian distribution. It equals 0 if the PDF has a Gaussian distribution (*mesokurtic*), is negative for a flatter distribution with a wider peak (*platykurtic*), and positive for distributions with a more acute peak (*leptokurtic*).

Similar to the spectral centroid and spectral spread, the *spectral skewness* interprets the spectrum as a distribution. It measures the symmetry of the distribution of the spectral magnitude values around their spectral centroid. It is defined by

$$v_{\mathrm{SSk}}(n) = \frac{\sum\limits_{k=0}^{\mathcal{K}/2} \left(k - v_{\mathrm{SC}}(n)\right)^3 \cdot |X(k,n)|}{v_{\mathrm{SS}}^3 \cdot \sum\limits_{k=0}^{\mathcal{K}/2} |X(k,n)|}. \tag{3.49}$$

The spectral skewness will result in a value close to zero for noisy signals and higher values for signals with their main energy in the low frequencies and is not defined for silent input (see Listing 3.8) and relies on the computation of spectral centroid and spectral spread.

Repo 2: FeatureSpectralSkewness.m
Repo 3: FeatureSpectralSkewness.py

```
% compute 'mean' and 'standard deviation'
mu_X = FeatureSpectralCentroid(X, f_s) * 2 / f_s * (size(X, 1)-1);
std_X = FeatureSpectralSpread(X, f_s) * 2 / f_s * (size(X, 1)-1);
tmp = repmat(0:size(X, 1)-1, size(X, 2), 1) - repmat(mu_X, size(X, 1), 1)';

vssk = sum((tmp.^3)'.*X)' ./ (std_X'.^3 .* sum(X, 1)'*size(X, 1));
```

Listing 3.8 Implementation of the *spectral skewness* computed from $|X(k,n)|$.

Figure 3.20 shows the spectral skewness for an example signal. During signal pauses, the spectral skewness drops since the spectral magnitudes are similar, while at times with high magnitudes at the fundamental frequency the magnitude spectrum is significantly skewed. The spectral skewness increases, for example in the region between 12 and 14 s due to the strong decrease of the higher harmonics compared to the lower harmonics.

The *spectral kurtosis* measures whether the distribution of the spectral magnitude values is shaped like a Gaussian distribution. It is defined by

$$v_{\mathrm{SK}}(n) = \frac{\sum\limits_{k=0}^{\mathcal{K}/2} \left(k - v_{\mathrm{SC}}(n)\right)^4 \cdot |X(k,n)|}{v_{\mathrm{SS}}^4 \cdot \sum\limits_{k=0}^{\mathcal{K}/2} |X(k,n)|} - 3. \tag{3.50}$$

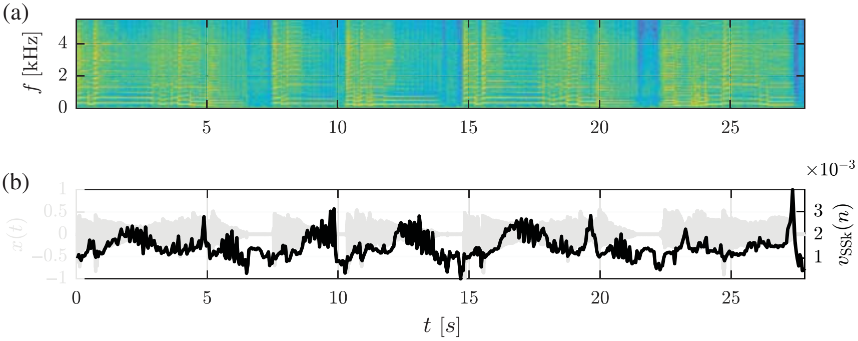


Figure 3.20 Spectrogram (a), waveform (b, background), and *spectral skewness* (b, foreground) of a saxophone signal.

Repo 1: plotFeature.m

The spectral kurtosis is not defined for silent input (see Listing 3.9) and relies on the computation of spectral centroid and spectral spread.

```
% compute mean and standard deviation
mu_X = FeatureSpectralCentroid(X, f_s) * 2 / f_s * (size(X, 1)-1);
std_X = FeatureSpectralSpread(X, f_s) * 2 / f_s * (size(X, 1)-1);
tmp = repmat(0:size(X, 1)-1, size(X, 2), 1) - repmat(mu_X, size(X, 1), 1)';

vsk = sum((tmp.^4)'.*X)' ./ (std_X'.^4 .* sum(X, 1)'*size(X, 1))-3;
```

Listing 3.9 Implementation of the *spectral kurtosis* computed from $|X(k, n)|$.

Repo 2: FeatureSpectralKurtosis.m
Repo 3: FeatureSpectralKurtosis.py

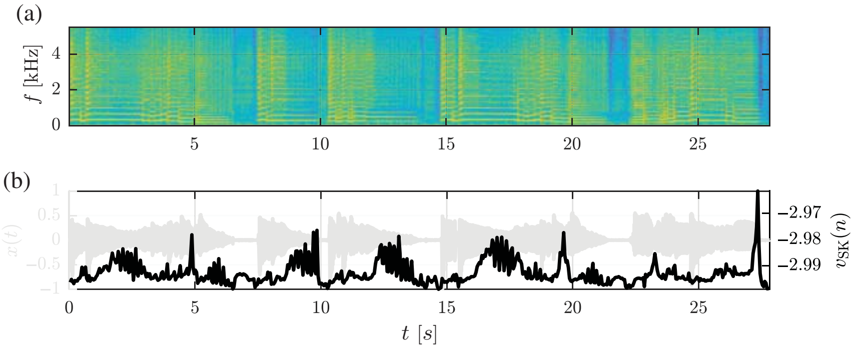
Figure 3.21 shows the spectral kurtosis for a saxophone signal. While during the notes high values can be observed, indicating a very peaked distribution, the spectral kurtosis drops significantly during pauses.

3.5.4 Spectral Rolloff

The *spectral rolloff* is a measure of the bandwidth of the analyzed block n of audio samples. The spectral rolloff $v_{\mathrm{SR}}(n)$ is defined as the frequency bin below which the accumulated magnitudes of the STFT $X(k, n)$ reach a certain percentage κ of the overall sum of magnitudes:

$$v_{\mathrm{SR}}(n) = k_{\mathrm{r}}\Bigg|_{\sum\limits_{k=0}^{k_{\mathrm{r}}}|X(k,n)|=\kappa\cdot\sum\limits_{k=0}^{\mathcal{K}/2}|X(k,n)|} \tag{3.51}$$

with common values for κ being 0.85 (85%) or 0.95 (95%). This can also be interpreted as the Quantile $Q_X(\kappa)$ (see Section 3.1.3.5) of the spectrum as a distribution.



Repo 1: plotFeature.m

Figure 3.21 Spectrogram (a), waveform (b, background), and *spectral kurtosis* (b, foreground) of a saxophone signal.

The result of the spectral rolloff is a bin index in the range $0 \leq v_{\mathrm{SR}}(n) \leq {}^{\mathcal{K}}\!/_{2}$. It can be converted either to Hertz with Eq. (3.36) or to a parameter range between zero and one by dividing it by the maximum frequency index ${}^{\mathcal{K}}\!/_{2}$. Low results indicate insignificant magnitude components at high frequencies and thus a low audio bandwidth. The spectral rolloff is not defined for silent input frames (see Listing 3.10).

```
% compute rolloff
afSum = sum(X, 1);
for n = 1:length(vsr)
    vsr(n) = find(cumsum(X(:, n)) >= kappa*afSum(n), 1)-1;
end

% convert from index to Hz
vsr = vsr / (size(X, 1)-1) * f_s / 2;
```

Repo 2: FeatureSpectralRolloff.m
Repo 3: FeatureSpectralRolloff.py

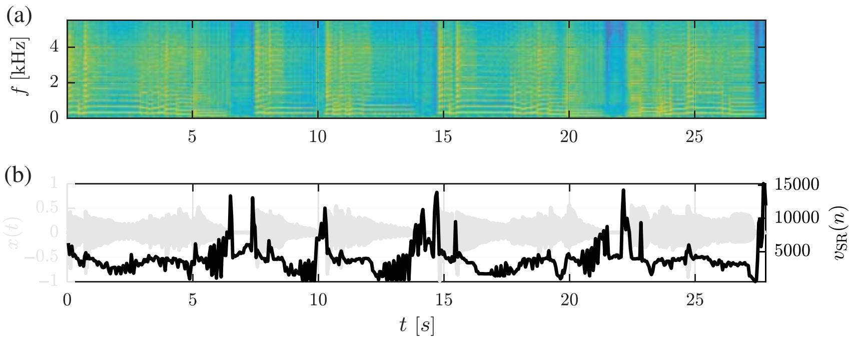
Listing 3.10 Implementation of the *spectral rolloff* computed from $|X(k,n)|$.

Figure 3.22 shows the spectral rolloff for an example signal. It is comparably low in the presence of a tone and higher – although somewhat erratic – during the noise-filled pauses.

Spectral bins at very low or very high frequencies may in many cases be considered to be unnecessary or unwanted for the analysis. Therefore, both sums in Eq. (3.51) may start and stop at predefined frequency boundaries $f_{\min}$, $f_{\max}$:

$$v_{\mathrm{SR},\Delta\mathrm{f}}(n) = k_{\mathrm{r}}\Bigg|_{\sum\limits_{k=k(f_{\min})}^{k_{\mathrm{r}}} |X(k,n)| = \kappa \cdot \sum\limits_{k=k(f_{\min})}^{k(f_{\max})} |X(k,n)|} . \tag{3.52}$$

It is also common to use the power spectrum instead of the magnitude spectrum.



Repo 1: plotFeature.m

Figure 3.22 Spectrogram (a), waveform (b, background), and *spectral rolloff* (b, foreground) of a saxophone signal.

3.5.5 Spectral Decrease

The *spectral decrease* estimates the steepness of the decrease of the spectral envelope over frequency. It is defined by [17]

$$v_{SD}(n) = \frac{\sum\limits_{k=1}^{\mathcal{K}/2} \frac{1}{k} \cdot \left(|X(k,n)| - |X(0,n)|\right)}{\sum\limits_{k=1}^{\mathcal{K}/2} |X(k,n)|}. \tag{3.53}$$

The result of the spectral decrease is a value $v_{SD}(n) \leq 1$. Low results indicate the concentration of the spectral energy at bin 0. The spectral decrease is not defined for silent input frames (see Listing 3.11).

```
% compute index vector
k = 0:size(X, 1)-1;
k(1) = 1;
kinv = 1 ./ k;

% compute slope
vsd = (kinv*(X-repmat(X(1, :), size(X, 1), 1))) ./ sum(X(2:end, :), 1);
```

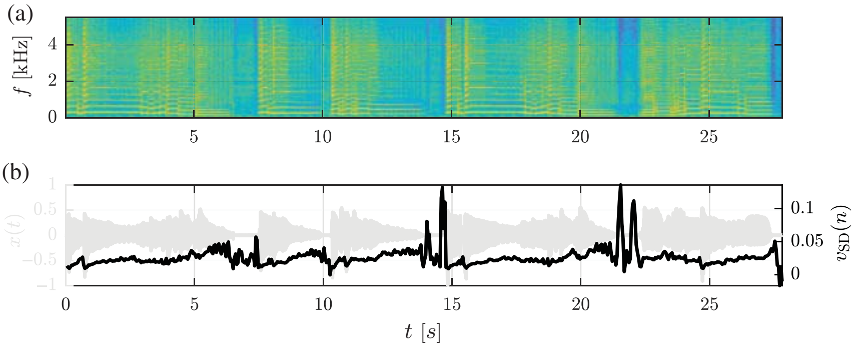
Repo 2: FeatureSpectralDecrease.m
Repo 3: FeatureSpectralDecrease.py

Listing 3.11 Implementation of the *spectral decrease* computed from $|X(k,n)|$.

Figure 3.23 shows the spectral decrease for an example signal. The difficulty to draw any conclusions from the plot except that the feature behaves erratically during the pauses explains why this feature is not frequently used.

3.5.6 Spectral Slope

The *spectral slope* is – similar to the spectral decrease – a measure of the slope of the spectral shape. It is calculated using a linear approximation of the magnitude



Repo 1: plotFeature.m

Figure 3.23 Spectrogram (a), waveform (b, background), and *spectral decrease* (b, foreground) of a saxophone signal.

spectrum; more specifically, a linear regression approach is used (see Appendix D). The spectral slope is estimated with the equation

$$v_{\mathrm{SSl}}(n) = \frac{\sum\limits_{k=0}^{\mathcal{K}/2} (k - \mu_k)(|X(k,n)| - \mu_{|X|})}{\sum\limits_{k=0}^{\mathcal{K}/2} (k - \mu_k)^2} \tag{3.54}$$

The result of the spectral slope depends on the amplitude range of the spectral magnitudes. Example code is shown in Listing 3.12.

```
% compute mean
mu_x = mean(abs(X), 1);

% compute index vector
kmu = (0:size(X, 1)-1) - size(X, 1)/2;

% compute slope
X = X - repmat(mu_x, size(X, 1), 1);
vssl = (kmu*X) / (kmu*kmu');
```

Repo 2: FeatureSpectralSlope.m
Repo 3: FeatureSpectralSlope.py

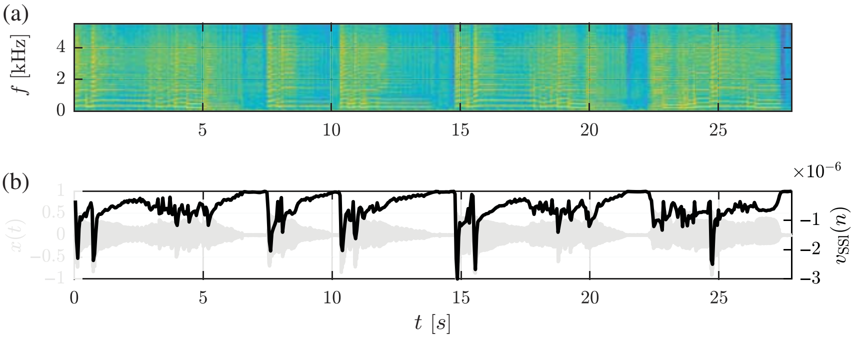
Listing 3.12 Implementation of the *spectral slope* computed from $|X(k,n)|$.

Figure 3.24 shows the spectral slope for an example signal. It increases with disappearing higher harmonics and is maximal for the noisy rests.

3.5.7 Mel Frequency Cepstral Coefficients

The *Mel Frequency Cepstral Coefficients (MFCCs)* are a compact description of the shape of the spectral envelope of an audio signal. The jth coefficient $v^j_{\mathrm{MFCC}}(n)$ can be calculated with

$$v^j_{\mathrm{MFCC}}(n) = \sum_{k'=1}^{\mathcal{K}'} \log\left(|X_{\mathrm{warp}}(k',n)|\right) \cdot \cos\left(j \cdot \left(k' - \frac{1}{2}\right) \frac{\pi}{\mathcal{K}'}\right) \tag{3.55}$$



Repo 1: plotFeature.m

Figure 3.24 Spectrogram (a), waveform (b, background), and *spectral slope* (b, foreground) of a saxophone signal.

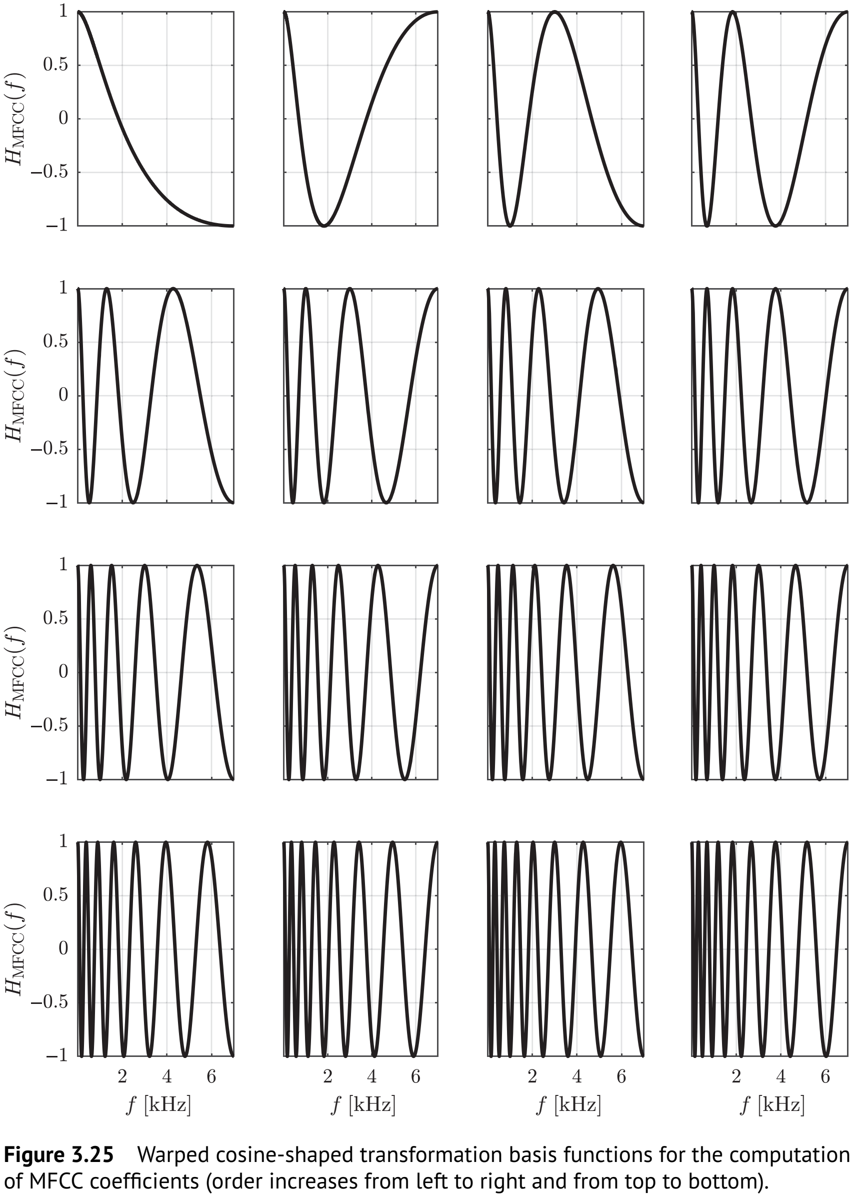
with $|X_{\mathrm{warp}}(k', n)|$ being the mel-warped magnitude spectrum at the signal block. The MFCC calculation is based on the following steps:

1. computation of the Mel-warped (see Section 7.1.1) spectrum with a bank of overlapping band-pass filters (see also Section 3.3.3),
2. taking the logarithm of the magnitude of each resulting band, and
3. calculating the *Discrete Cosine Transform (DCT)* on the resulting bands. The DCT is similar to the real (cosine) part of an FT.

MFCCs have been widely used in the field of speech signal processing since their introduction in 1980 [38] and have been found to be useful in music signal processing applications as well [39–42]. In the context of audio signal classification, it has been shown that a small subset of the resulting MFCCs as shown in Figure 3.26 already contains the principal information [43, 44] – in most cases the number of used MFCCs varies in the range from 4 to 20. Nowadays, the MFCCs are probably the most frequently used audio features for baseline systems due to their proven robustness and usefulness over a wide range of tasks.

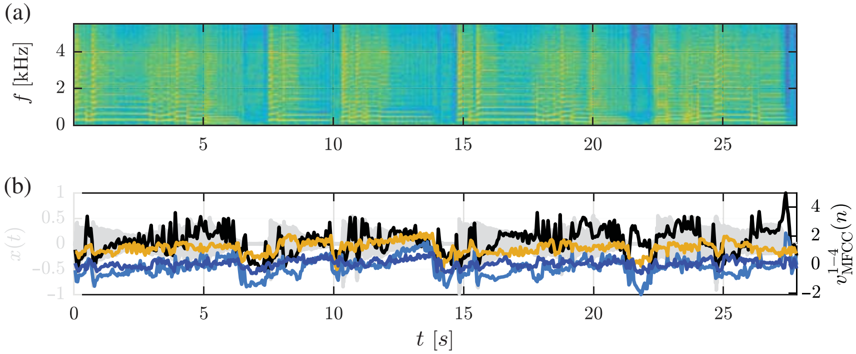
The calculation is closely related to the calculation of the cepstrum as introduced in Section 7.3.3.6 as it transforms a spectral representation after applying a logarithm to it. The main difference to the standard cepstrum is the use of a warped nonlinear frequency scale (the Mel scale, see Section 7.1.1) to model the nonlinear human perception of frequency and the use of the DCT instead of a DFT.

The Mel-warped basis functions for the DCT are displayed in Figure 3.25. The Mel warping of the spectrum frequently leads to the claim that the MFCCs are "perceptual" features. This is only partly true as there is no psychoacoustic evidence to motivate the application of the DCT. Furthermore, there is no direct correlation between the MFCCs and known perceptual dimensions.



Repo 1: plotMfccMelDct.m

Figure 3.25 Warped cosine-shaped transformation basis functions for the computation of MFCC coefficients (order increases from left to right and from top to bottom).



Repo 1: plotFeature.m

Figure 3.26 Spectrogram (a) and Mel frequency cepstral coefficients 1–4 (b) of a saxophone signal.

The result of the MFCCs depends on the amplitude range of the spectral power. The zeroth Mel Frequency Cepstral Coefficient (MFCC) $v^0_{\mathrm{MFCC}}(n)$ is usually ignored as it has no relevance in describing the timbre. It is highly correlated to the energy in decibel. The first four coefficients are shown in Figure 3.26. Despite their proven usefulness, it is difficult to identify nontrivial relationships to the input signal.

The MFCCs are not defined for silence as input signal. An excerpt of the source code to extract the MFCCs is shown in Listing 3.13.

```
% create filters
H = ToolMfccFb(size(X, 1), f_s);

% create transformation matrix
T = GenerateDctMatrix_I (size(H, 1), iNumCoeffs);

for n = 1:size(X,2)
    % compute the mel spectrum
    X_Mel = log10(H * X(:, n) + 1e-20);

    % calculate the mfccs
    vmfcc(:, n) = T * X_Mel;
end
```

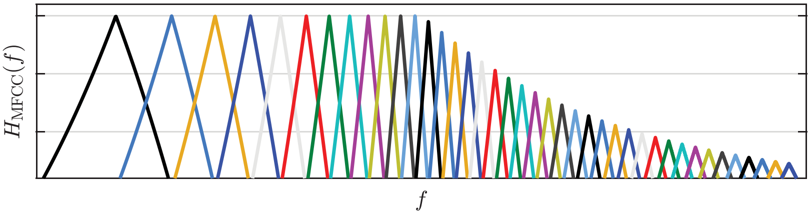
Repo 2: FeatureSpectralMfccs.m
Repo 3: FeatureSpectralMfccs.py

Listing 3.13 Implementation of the *spectral mfccs* computed from $|X(k,n)|$.

The differences between MFCC implementations can be found mainly in the computation of the mel-warped spectrum, i.e. in number, location, and normalization of the filters. Table 3.1 shows the differences between the three most popular MFCC implementations, the original introduced by Davis and Mermelstein (DM) [38], the implementation in the *HMM Toolkit (HTK)* software [45], and the implementation in Slaney's *Auditory Toolbox* (SAT) [46].

Table 3.1 Properties of three popular MFCC implementations, Davis and Mermelstein (DM), the software HTK, and Slaney's Auditory Toolbox (SAT).

Property	*DM*	*HTK*	*SAT*
Num. filters	20	24	40
Mel scale	lin/log	log	lin/log
Freq. range	[100; 4000]	[100; 4000]	[200; 6400]
Normalization	Equal height	Equal height	Equal area



Repo 1: plotMfccFilterbank.m

Figure 3.27 Magnitude transfer function of the filterbank for MFCC computation as used in Slaney's *Auditory Toolbox*. Source: Modified from Slaney [46].

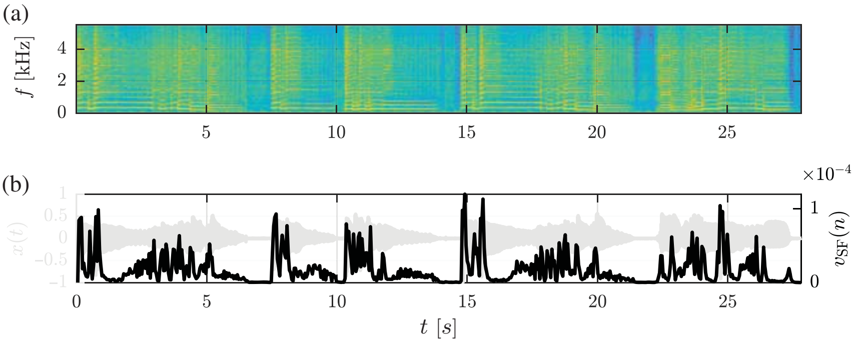
Section 7.1.1 lists the typical Mel scale models used for the nonlinear frequency warping. It is also possible to use other filter shapes or to compute MFCCs directly from the power spectrum by using warped cosine basis functions as shown in Figure 3.25 [47]. Figure 3.27 shows the triangular filter shapes as used in the Slaney's *Auditory Toolbox*.

3.5.8 Spectral Flux

The *spectral flux* is different from many other features described here in that it measures the amount of change of the spectral shape instead of describing the spectral shape itself. It is defined as the average difference between consecutive STFT frames:

$$v_{\mathrm{SF}}(n) = \frac{\sqrt{\sum\limits_{k=0}^{\mathcal{K}/2} \left(|X(k,n)| - |X(k,n-1)|\right)^2}}{\mathcal{K}/2 + 1}. \tag{3.56}$$

In some ways, the spectral flux can be seen as related to the sensation of *roughness* which, according to Zwicker and Fastl, is driven by a quasiperiodic change or a modulation in the excitation pattern levels [35].



Repo 1: plotFeature.m

Figure 3.28 Spectrogram (a), waveform (b, background), and *spectral flux* (b, foreground) of a saxophone signal.

The result of the spectral flux is a value within the range $0 \leq v_{\mathrm{SF}}(n) \leq A$ with A representing the maximum possible spectral magnitude. Thus, its output range depends on the normalization of the audio signal and the normalization of the Fourier transform. Low results indicate steady-state input signals or low-input levels. The source code for the computation of the spectral flux can be found in Listing 3.14.

Figure 3.28 shows the spectral flux for an example signal. It is low during the stationary parts of the signal, such as during a note or a pause, and spikes at pitch changes and at the beginning of a new note.

Repo 2: FeatureSpectralFlux.m
Repo 3: FeatureSpectralFlux.py

```
% difference spectrum (set first diff to zero)
afDeltaX = diff([X(:, 1), X], 1, 2);

% flux
vsf = sqrt(sum(afDeltaX.^2)) / size(X, 1);
```

Listing 3.14 Implementation of the *spectral flux* computed from $|X(k, n)|$.

The definition of the spectral flux above is the Euclidean distance of the two spectra. The distance norm can also be generalized:

$$v_{\mathrm{SF}}(n, \beta) = \frac{\sqrt[\beta]{\sum\limits_{k=0}^{\mathcal{K}/2} \left(|X(k, n)| - |X(k, n-1)|\right)^{\beta}}}{\mathcal{K}/2 + 1}. \tag{3.57}$$

Typical values for β range between [0.25, 3], with $\beta = 1$ resulting in the Manhattan distance, see Eq. (4.6), and $\beta = 2$ resulting in the Euclidean distance, see Eq. (4.5).

In some applications such as note onset detection, only an increase in spectral energy is of interest; in these cases, the difference magnitude spectrum can be computed:

$$\Delta X(k,n) = |X(k,n)| - |X(k,n-1)| \tag{3.58}$$

and all negative differences $\Delta X(k,n) < 0$ will be set to zero before summation while positive differences will be left unaltered. This is called *Half-Wave Rectification (HWR)*. Mathematically, the HWR of a signal x can be written as

$$\mathrm{HWR}(x) = \frac{x + |x|}{2}. \tag{3.59}$$

Alternative approaches can be used to derive a measure of spectral change. Examples are the computation of the standard deviation of the difference magnitude spectrum $\Delta X(k,n)$:

$$v_{\mathrm{SF},\sigma}(n) = \sqrt{\frac{2}{\mathcal{K}+2}\sum_{k=0}^{\mathcal{K}/2}\left(\Delta X(k,n) - \mu_{\Delta X}\right)^2}, \tag{3.60}$$

and the computation of the logarithmic difference. The latter has the advantage of making the resulting feature independent of magnitude scaling but the disadvantage of being undefined for zero-mean frames:

$$v_{\mathrm{SF,log}}(n) = \frac{2}{\mathcal{K}+2}\sum_{k=0}^{\mathcal{K}/2}\log_2\left(\frac{|X(k,n)|}{|X(k,n-1)|}\right). \tag{3.61}$$

3.5.9 Spectral Crest Factor

The *spectral crest factor* gives an estimate on “how sinusoidal” a spectrum is. It is a simple measure of *tonalness* in the sense that it crudely estimates the amount of *tonal* components in the signal as opposed to noisy components.[3]

The spectral crest factor is the ratio of maximum of the magnitude spectrum and the sum of this magnitude spectrum. It is, therefore, defined by

$$v_{\mathrm{Tsc}}(n) = \frac{\max\limits_{0\le k\le \mathcal{K}/2} |X(k,n)|}{\sum\limits_{k=0}^{\mathcal{K}/2} |X(k,n)|}. \tag{3.62}$$

The result of the spectral crest factor is a value between $2/(\mathcal{K}+2) \le v_{\mathrm{Tsc}}(n) \le 1$. Low results indicate a flat magnitude spectrum and high results indicate a sinusoidal. The spectral crest factor is not defined for audio blocks with no spectral energy (see Listing 3.15).

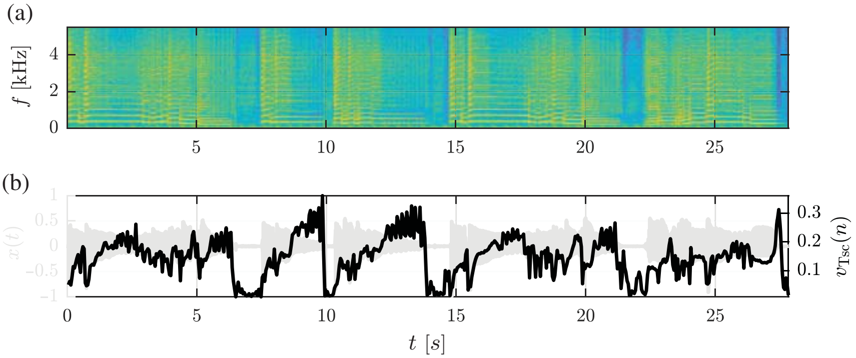
---

[3] The somewhat uncommon term *tonalness* is used here to distinguish this measure from the musical term *tonality* which describes a specific harmonic or key context.

```
% get maximum
vtsc = max(X, [], 1) ./ sum(X, 1);

% avoid NaN for silence frames
vtsc (sum(X, 1) == 0) = 0;
```

Repo 2: FeatureSpectralCrestFactor.m
Repo 3: FeatureSpectralCrestFactor.py

Listing 3.15 Implementation of the *spectral crestfactor* computed from $|X(k,n)|$.



Repo 1: plotFeature.m

Figure 3.29 Spectrogram (a), waveform (b, background), and *spectral crestfactor* (b, foreground) of a saxophone signal.

Figure 3.29 shows the spectral crest factor for an example signal. It is low for noisy parts during the pauses and higher during the tonal passages. With decreasing amplitude of higher harmonics, the spectral crest factor increases as the spectral energy is more and more concentrated at a single spectral bin.

In some implementations, the denominator of Eq. (3.62) can also be the arithmetic mean of the magnitude spectrum. This scales the range of the spectral crest factor to $1 \leq v_{\mathrm{Tsc}}(n) \leq {}^{(\mathcal{K}+2)}\!/_{2}$.

3.5.10 Spectral Flatness

The *spectral flatness* is the ratio of geometric mean and arithmetic mean of the magnitude spectrum. It is defined by [48]

$$v_{\mathrm{Tf}}(n) = \frac{\sqrt[\mathcal{K}/2]{\prod\limits_{k=0}^{\mathcal{K}/2-1} |X(k,n)|}}{{}^{2}\!/_{\mathcal{K}} \cdot \sum\limits_{k=0}^{\mathcal{K}/2-1} |X(k,n)|} = \frac{\exp\left({}^{2}\!/_{\mathcal{K}} \cdot \sum\limits_{k=0}^{\mathcal{K}/2-1} \log\left(|X(k,n)|\right)\right)}{{}^{2}\!/_{\mathcal{K}} \cdot \sum\limits_{k=0}^{\mathcal{K}/2-1} |X(k,n)|}. \tag{3.63}$$

The latter formulation uses the arithmetic mean of the logarithmic magnitude spectrum in the numerator in order to avoid problems with numerical accuracy.

The result of the spectral flatness is a value larger than 0. The upper limit depends on the maximum spectral magnitude. Low results hint toward a non-flat – possibly a tonal – spectrum, while high results indicate a flat (or noisy) spectrum. The spectral flatness is thus a measure of noisiness as opposed to tonalness. Note that as soon as the magnitude at one individual bin equals 0, v_{Tf} will be zero as well. The implementation of the spectral flatness is shown in Listing 3.16. The behavior of the spectral flatness at pauses in the input signal requires special consideration as it is not defined for silence and will be comparably large for (low-level) noise.

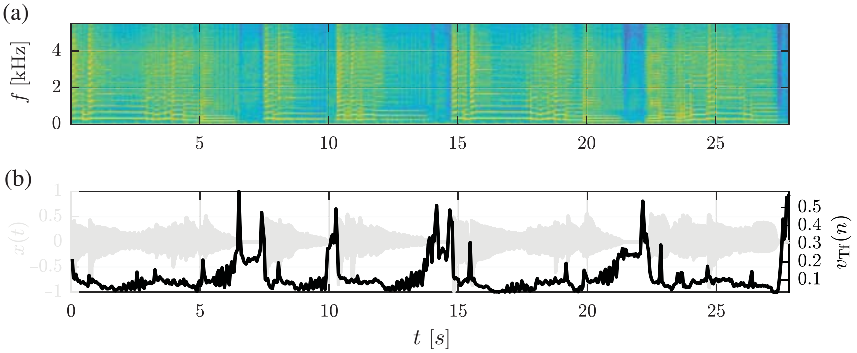
```
% replace product sum with log/exp
XLog = log(X + 1e-20);
vtf = exp(mean(XLog, 1)) ./ (mean(X, 1));

% avoid NaN for silence frames
vtf (min(X, [], 1) == 0) = 0;
```

Repo 2: FeatureSpectralFlatness.m
Repo 3: FeatureSpectralFlatness.py

Listing 3.16 Implementation of the *spectral flatness* computed from $|X(k,n)|$.

Figure 3.30 shows the spectral flatness for an example signal. It is low during tonal passages, high in noisy pauses, and produces spikes at transients.

It is common to use the power spectrum instead of the magnitude spectrum in order to emphasize peaks. To avoid problems with individual zero magnitudes negatively impacting the overall result, the magnitude spectrum can be smoothed.



Repo 1: plotFeature.m

Figure 3.30 Spectrogram (a), waveform (b, background), and *spectral flatness* (b, foreground) of a saxophone signal.

One typical approach is to compute the arithmetic mean of a group of neighboring spectral coefficients, basically applying an MA filter to the magnitude spectrum. However, the length of the filter might also increase with frequency to take into account the lower-frequency resolution of the human ear at high frequencies. In many cases, more useful information can be gathered if the spectral flatness calculation takes only magnitudes within a predefined frequency range into account, as opposed to computing it from the whole spectrum. The MPEG-7 standard recommends a frequency range of from 250 Hz to 16 kHz, divided into 24 slightly overlapping frequency bands with quarter-octave bandwidth [16]. Since the spectral flatness is then computed for each individual frequency band, the result per STFT is a vector of spectral flatness results.

3.5.11 Tonal Power Ratio

Another way to compute the tonalness of a spectrum is to compute the ratio of the tonal power $E_\mathrm{T}(n)$ to the overall power:

$$v_\mathrm{Tpr} = \frac{E_\mathrm{T}(n)}{\sum\limits_{i=0}^{\mathcal{K}/2} |X(k,n)|^2}. \tag{3.64}$$

This reduces the problem to the estimation of the power of the tonal components. A simple approximation to estimating the tonal energy is summing the power at all bins k which

- are a local maximum: $|X(k-1,n)|^2 \leq |X(k,n)|^2 \geq |X(k+1,n)|^2$ and
- have power above a threshold G_T.

The result of the tonal power ratio is a value between $0 \leq v_\mathrm{Tpr} \leq 1$. Low results hint toward a flat (noisy) spectrum or a block with low-input level while high results indicate a tonal spectrum. The tonal power ratio is not defined for audio blocks with no spectral energy (see Listing 3.17).

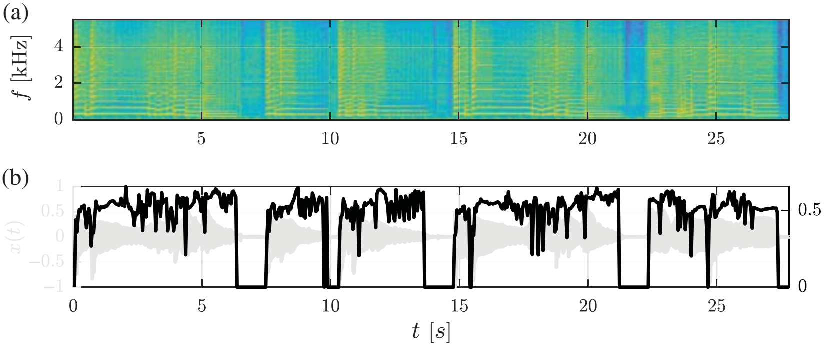
Figure 3.31 shows the tonal power ratio for an example signal. It is zero in noisy pauses, high for tonal passages and, in the case of this simple saxophone signal, drops distinctively at the initial transients at note beginnings.

There are many proposed approaches for identifying tonal components in the spectral domain, as the range of applications which can benefit from a reliable tonalness detector spans: (i) psycho-acoustic models in perceptual audio encoders that can be optimized with a more accurate estimation of the signal-to-mask ratio, (i) source separation algorithms and analysis/synthesis systems such as phase vocoders which may be improved by treating tonal and noisy components separately, and (i) pitch-based analysis systems such as key detection and chord recognition and ultimately music transcription systems for which nontonal

```
fSum = sum(X, 1);

for n = 1:size(X, 2)
    if (fSum(n) == 0)
        % do nothing for 0-blocks
        continue;
    end
    % find local maxima
    [afPeaks] = findpeaks(X(:, n));

    % find peaks above the threshold
    k_peak = find(afPeaks > G_T);

    % calculate the ratio
    vtpr(n) = sum(afPeaks(k_peak)) / fSum(n);
end
```

Repo 2: FeatureSpectralTonalPowerRatio.m
Repo 3: FeatureSpectralTonalPowerRatio.py

Listing 3.17 Implementation of the *spectral tonalpowerratio* computed from $|X(k, n)|$.



Repo 1: plotFeature.m

Figure 3.31 Spectrogram (a), waveform (b, background), and *spectral tonalpowerratio* (b, foreground) of a saxophone signal.

information usually should be removed as irrelevant. Similar to the simple approach mentioned above, Parsons identified peaks by finding local maxima in the magnitude spectrum – a first processing step that can be found in practically every publication dealing with the detection of tonal bins – and used the peak's symmetry, its proximity to the next peak as well as the continuity of the frequency bin's phase for detecting "peak overlaps," i.e. bins with supposedly two or more influencing sinusoidals [49]. Terhardt extended the concept of detecting local maxima by expecting more distant bins (specifically the bins with a distance of 2 and 3) to be a certain level lower than the maximum itself [50]. Serra proposed a measure of "peakiness" of local maxima by comparing the bin magnitude with the surrounding local minima; he also discarded peaks outside of a predefined frequency and magnitude range [51]. An amplitude-based measure computing the correlation function between the magnitude spectrum and the shifted spectrum of the used window function has been presented by Peeters and Rodet [52] as well as Lagrange [53]. They also utilized a phase-derived measure comparing the bin frequency of a peak with its reassigned (instantaneous) frequency (see Sec. 17.6). In addition to a local maximum feature similar to Terhardt's, Every proposed to discard peaks below an adaptive threshold computed from the low-pass filtered magnitude spectrum [54]. Röbel et al. presented a set of features to classify spectral peaks into being sinusoidal or nonsinusoidal [55]. These features included the deviation of the bin frequency and its reassigned frequency, the peak's energy location according to its group delay, as well as the bandwidth of a spectral peak.

The publications presented above make a binary decision for a spectral bin being tonal or not; Kulesza and Czyzewski proposed an algorithm which aims at estimating the likelihood of a bin's tonalness [56]. They refer to this as a scoring classifier and use a so-called "peakiness" feature similar to Serra's, a frequency stability criterion for detected peaks, and a phase-based frequency coherence over subsequent blocks of the STFT. Their approach combines several features and uses a combination of heuristics and both binary and nonbinary features to compute the resulting likelihood. Similarly, Kraft and Lerch extract multiple features per frequency local maximum, including its "peakiness," its saliency (computed by adaptive thresholding), and a "frequency coherence" measure based on the distance of the bin frequency and its instantaneous frequency [57]. The individual features are then weighted with a Gaussian function and aggregated for an overall tonalness measure.

3.5.12 Maximum of Autocorrelation Function

The Autocorrelation Function (ACF) of a time signal yields local maxima where the ACF lag matches the wavelengths of the signal-inherent periodicities (see Appendix A.3.2). The less periodic and therefore less tonal the signal is, the lower is the value of such maxima. The absolute value of the overall *ACF maximum* is therefore a simple estimate of the signal's tonalness:

$$v_{\mathrm{Ta}}(n) = \max_{\eta_{\mathrm{start}} \leq \eta \leq \mathcal{K}-1} |r_{xx}(\eta, n)|. \tag{3.65}$$

Values in the main lobe of the ACF around lag $\eta = 0$ have to be discarded to ensure useful results. Different approaches can be used to ignore the main lobe, i.e. to adjust η_{start}:

- *Minimum lag*: Assuming that a maximum of interest will not be found at high frequencies (i.e. at small lags and period lengths), the search for the maximum can be started at a predefined lag, ignoring values at smaller lags. The lower the expected maximum frequency is, the larger the minimum lag can be. Depending on the task at hand and the sample rate, the maximum frequency might be too high to correspond to a reasonably large minimum lag. At a sample rate of 48 kHz, for example, a frequency of 9.6 kHz corresponds to a lag of 5 samples, a frequency of 4.8 kHz to a lag of 10 samples, and a frequency of 1920 Hz to a lag of 25 samples.

- *Minimum magnitude threshold*: Set lag η_{start} to the smallest lag at which r_{xx} crosses a predefined threshold G_r

$$\eta_\text{r} = \operatorname*{argmin}_{0\leq\eta\leq\mathcal{K}-1} \left(r_{xx}(\eta) < G_\text{r}\right). \tag{3.66}$$

  Note that theoretically, the threshold might never be crossed; this case has to be considered in the implementation.
- *Search range from the first local minimum*: Only consider maxima at lags larger than the lag of the "first" local minimum. The idea is to avoid the detection of "insignificant" local maxima in the main lobe around lag $\eta = 0$, but depending on the signal, a local minimum might be detected at a very low lag.

Practically, a combination of these approaches plus additional, problem-specific constraints can be used to reliably adjust η_{start}.

The result is a value between $0 \leq v_{\text{Ta}}(n) \leq 1$. This ACF-based feature will work best for monophonic signals or signals with a limited number of fundamental frequencies. Low results indicate a nonperiodic signal and high result values indicate a periodic signal (see Listing 3.18).

Figure 3.32 shows the ACF maximum for an example signal. There is the tendency of low-feature values at noisy pause segments and higher values for tonal segments.

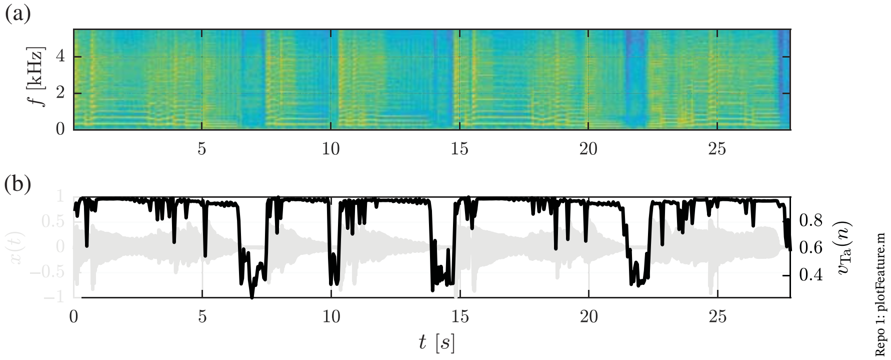


Figure 3.32 Spectrogram (a), waveform (b, background), and feature *maxacf* (b, foreground) of a saxophone signal.

```
    for n = 1:iNumOfBlocks
        eta_min = ceil(f_s / f_max);

        % calculate the acf
        afCorr = xcorr(x_b(n, :), ’coeff’);
        afCorr = afCorr((ceil((length(afCorr)/2))+1):end);

        % ignore values until threshold was crossed
        eta_tmp = find (afCorr < fMinThresh, 1);
        if (~isempty(eta_tmp))
            eta_min = max(eta_min, eta_tmp);
        end

        % only take into account values after the first minimum
        afDeltaCorr = diff(afCorr);
        eta_tmp = find(afDeltaCorr > 0, 1);
        if (~isempty(eta_tmp))
            eta_min = max(eta_min, eta_tmp);
        end

        % find the maximum in the computed range
        vta(n) = max(abs(afCorr(1+eta_min:end)));
    end
```

Repo 2: FeatureTimeMaxAcf.m
Repo 3: FeatureTimeMaxAcf.py

Listing 3.18 Implementation of *maxacf* from the time signal $x(i)$.

3.5.13 Zero Crossing Rate

The number of changes of sign in consecutive blocks of audio samples – the *zero crossing rate* – is a low-level feature that has been used for decades in speech and audio analysis due to its simple calculation:

$$v_{\mathrm{ZC}}(n) = \frac{1}{2\cdot\mathcal{K}} \sum_{i=i_{\mathrm{s}}(n)}^{i_{\mathrm{e}}(n)} \left|\mathrm{sign}\left[x(i)\right] - \mathrm{sign}\left[x(i-1)\right]\right| \tag{3.67}$$

with the sign function being defined by

$$\mathrm{sign}\left[x(k)\right] = \begin{cases} 1, & \text{if } x(i) > 0 \\ 0, & \text{if } x(i) = 0\ . \\ -1, & \text{if } x(i) < 0 \end{cases} \tag{3.68}$$

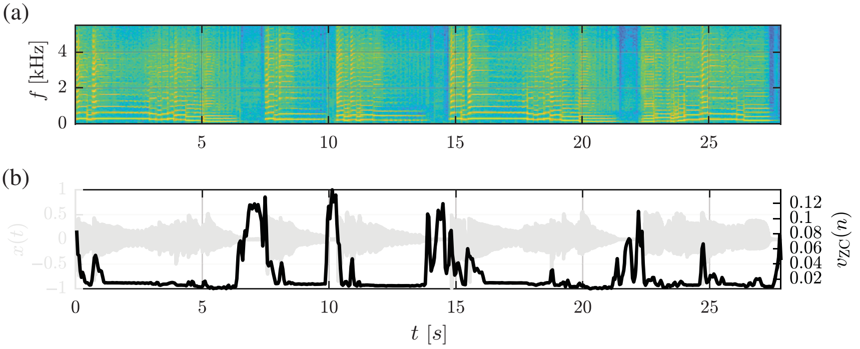
If $x(i-1)$ does not exist, it can be either initialized to zero or the first sample can be ignored.

The output is a value in the range of $0 \le v_{\mathrm{ZC}}(n) \le 1$. The more often the signal changes its sign, the more noisy or high-frequency content can be assumed to be in the signal. Furthermore, the more the zero crossing rate varies over blocks, the less periodic the signal can be assumed to be. The concept of the zero crossing rate is based on the assumption that the input signal has an arithmetic mean of approximately 0. An implementation of the zero crossing rate is shown in Listing 3.19.

Repo 2: FeatureTimeZeroCrossingRate.m
Repo 3: FeatureTimeZeroCrossingRate.py

```
for n = 1:iNumOfBlocks
    % compute the zero crossing rate
    vzc(n)   = 0.5 * mean(abs(diff(sign(x_b(n, :)))));
end
```

Listing 3.19 Implementation of *zerocrossingrate* from the time signal $x(i)$.



Repo 1: plotFeature.m

Figure 3.33 Spectrogram (a), waveform (b, background), and feature *zerocrossingrate* (b, foreground) of a saxophone signal.

Figure 3.33 shows the zero crossing rate for an example signal. As mentioned above, it is high for noisy parts and low for tonal parts. Its usability for fundamental frequency detection (see Section 7.3.3.1) is indicated by the long constant values for constant pitches.

The zero crossing rate has been used for both measuring the noisiness of a signal and estimating its fundamental frequency by assuming a sinusoidal input signal and then relating the number of zero crossings directly to the fundamental frequency. To improve the robustness of such attempts, the input signal can be low-pass filtered to suppress high-frequency content. The cut-off frequency of the low-pass filter should be chosen as low as the highest expected fundamental frequency to ensure maximum suppression of high-frequency content.

3.6 Learned Features

The low-level features described above are only a subset of the vast set of handcrafted and custom-designed features that have been successfully applied to audio analysis tasks for many years. As classifiers grew more powerful over time, researchers started to identify problems with this expert-driven approach. Humphrey et al. argued, for example, that "hand-crafted feature design is neither scalable nor sustainable" and that "shallow processing architectures struggle to describe the latent complexity of real-world phenomena" and called for data-driven feature design, known as *feature learning* [58].

The advantage of using features learned from the data itself is that an algorithm might be able to identify and extract hidden relevant attributes in the data which might not be sufficiently represented in low-level features. This might be even true for randomly projected features: a randomly initialized untrained neural network may have the power to project the input data into a feature space that enables successful inference [59]. Unsurprisingly, however, a random projection will often be outperformed by a projection learned from relevant data. Feature learning methods can be categorized into unsupervised and supervised approaches, depending on whether unlabeled data or labeled data are used for training.

Unsupervised approaches to feature learning have been extensively explored in the late 2000s and early 2010s. Sparse coding, which attempts to learn a high-dimensional sparse representation of audio data via a dictionary learning approach, gained some popularity by learning features from spectrogram-like representations of the data [60–62]. Other researchers proposed to learn features from spectrograms with a K-Means algorithm [63, 64]. These and other methods for feature learning have also been evaluated in comparison [65, 66], often with the result that deep neural architectures [67] generally seem to exhibit superior performance. Typical architectures include Deep Belief Networks [68–70] or auto-encoders [71].

The current trend seems to go toward supervised approaches for feature learning. Due to the integration of feature extraction and inference in modern networks, the term feature learning is not frequently used anymore and is mostly replaced with the term *representation learning*, although the concept of learning a compressed, meaningful representation from the audio data remains unchanged. One aspect related to the concept of feature learning is *transfer learning* [72]. In this case, a network is trained on one task and subsequently the learned representation is used for a different albeit related task. Choi et al., for example trained a network for an auto-tagging task and then successfully used the embedded representation of the trained network as a feature input to a classifier for music genre classification [73]. Other widely used audio features (also "embeddings" or "representations"), all extracted by very deep architectures trained on large amounts of data, are VGGish [74], L3 [75], and musicnn [76]. These audio features have been successfully applied to a variety of tasks ranging from audio event classification [77] to music instrument classification [78].

Despite the often superior performance of features learned by deep architectures, there exist potential drawbacks and arguments against them. First of all, if only small amounts of data are available for training, a general representation cannot be learned. Second, learned features are often entangled and hard to interpret and thus not useful for an analysis of important factors and an interpretation or explanation of the classifier behavior. Last but not least, Krig points out that a learned feature is not necessarily as impartial as the data-driven approach might imply, as "DNN architectures and training methods are very handcrafted, and rely on several ad hoc design assumptions" [79].

3.7 Feature PostProcessing

The result of the feature extraction process is a series of feature values that can – dependent on the use case – be processed, transformed, and selected.

It is common to compute a large number of features. All these feature time series can be combined in a feature matrix:

$$\begin{aligned} \boldsymbol{V} &= \left[v(0)\ v(1)\ \cdots\ v(\mathcal{N}-1)\right] \\ &= \begin{bmatrix} v_0(0) & v_0(1) & \cdots & v_0(\mathcal{N}-1) \\ v_1(0) & v_1(1) & \cdots & v_1(\mathcal{N}-1) \\ \vdots & \vdots & \ddots & \vdots \\ v_{\mathcal{F}-1}(0) & v_{\mathcal{F}-1}(1) & \cdots & v_{\mathcal{F}-1}(\mathcal{N}-1) \end{bmatrix} \end{aligned} \tag{3.69}$$

with the number of rows being the number of features $\mathcal{F}$ and the number of columns being the number of blocks $\mathcal{N}$. Each vector $v(n)$ consists of $\mathcal{F}$ feature values at block n and will be referred to as an *observation*.

3.7.1 Derived Features

In some cases, it can be beneficial to process the extracted features to create an extended set of features. The resulting new "derived" features do not have to replace the original features; they can be added as additional features. The most commonly applied processing options are filters, especially both high-pass filters and low-pass filters. Filtering the feature series might help to remove irrelevant information. For example, in some cases, the detection of (sudden) changes of feature values is of special interest as it may mark the start or end of important segments such as note onsets and structural boundaries. A simple way of analyzing these changes is to compute the difference between consecutive feature results with a simple high-pass filter (this would be called *derivative* if it were a continuous function):

$$v_{j,\Delta}(n) = v_j(n) - v_j(n-1). \tag{3.70}$$

The resulting series $v_{j,\Delta}(n)$ is either one value shorter than the corresponding series $v_j(n)$ or an appropriate initialization for $v_j(-1)$ has to be defined.

The sequence $v_j(n)$ can also be smoothed out with a low-pass filter. This allows to identify long-term variations of the feature more easily. Two commonly used smoothing filters are the MA filter as introduced in Appendix A.2.6.1 or the single-pole low-pass filter as introduced in Appendix A.2.6.2. It is generally beneficial if the used filter has a zero phase or linear phase response in order to retain correct timing properties. An MA filter has linear phase response, but if any Infinite Impulse Response (IIR) filter should be used, it can be applied twice, forward and backward on the series, to produce the low-pass filtered series $v_{j,\mathrm{LP}}(n)$ with no phase shift (see Appendix A.2.7).

Other derived features might involve computing linear or nonlinear combinations of the existing features. The usage of such derived features is frequently met with only limited success as most of the information of interest can already be found in the original features. Furthermore, increasing the number of features and thus the dimensionality of the observations can potentially have negative impact on the classifier performance.

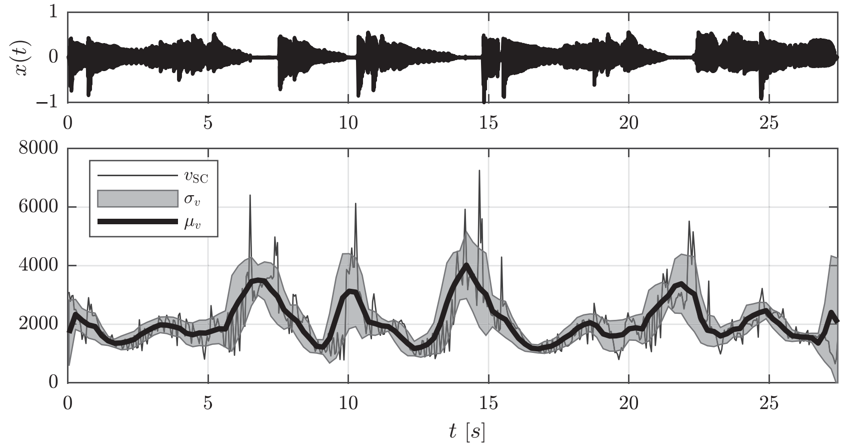
3.7.2 Feature Aggregation

The feature matrix $\boldsymbol{V}$ contains the feature value per block and each column might be treated as an observation vector to feed to the inference system. Practically, however, the comparably high-time resolution leads to large variations of the feature values which complicates any inference such as a classification. Therefore, it is common to combine multiple neighboring blocks into windows (sometimes referred to as *texture windows*), which might overlap in time. Each window contains a submatrix; in order to convert it to a single vector, statistical descriptors can be computed for each feature $v_j(n)$ per window. The most common statistical aggregations are the arithmetic mean and the standard deviation, although any measure presented in Section 3.1.3 could be used. In case of $\mathcal{F}$ features and a feature aggregation via the arithmetic mean and the standard deviation for each feature, the resulting length of the new observation vector fed to the inference system will be $2\mathcal{F}$. Figure 3.34 shows the aggregation of one texture window of length 4 using the arithmetic mean and the standard deviation.

The resulting observations are sometimes referred to as *subfeatures*. A relatively large number of possible meaningful subfeatures in the context of musical genre classification was evaluated by Mörchen et al. [80]. Note that the process can also be applied multiple times, leading to a hierarchical feature aggregation with decreasing time resolution as the hierarchy level increases. This hierarchical feature aggregation is somewhat similar to the repeated use of pooling layers often found in Convolutional Neural Networks CNNs.

$$\begin{bmatrix} \underbrace{\begin{matrix} v_0(0) & v_0(1) & v_0(2) & v_0(3) \\ v_1(0) & v_1(1) & v_1(2) & v_1(3) \\ \vdots & \vdots & \vdots & \vdots \\ v_{\mathcal{F}-1}(0) & v_{\mathcal{F}-1}(1) & v_{\mathcal{F}-1}(2) & v_{\mathcal{F}-1}(3) \end{matrix}}_{\begin{bmatrix} \mu_0(0) \\ \sigma_0(0) \\ \mu_1(0) \\ \sigma_1(0) \\ \vdots \\ \mu_{\mathcal{F}-1}(0) \\ \sigma_{\mathcal{F}-1}(0) \end{bmatrix}} & \begin{matrix} v_0(4) \\ v_1(4) \\ \vdots \\ v_{\mathcal{F}-1}(4) \end{matrix} \end{bmatrix}$$

Figure 3.34 Example aggregation with a texture window length of 4 with the arithmetic mean μ and the standard deviation σ.



Repo 1: plotTextureWindow.m

Figure 3.35 Feature aggregation with texture windows: example audio input (top), extracted feature (middle), and aggregated mean and standard deviation with overlapping texture windows (bottom).

The typical length of the texture window depends on the task; it can vary from values below 0.5 s for quasireal time inference or 15–30 s for various audio classification tasks to up to the song length. Figure 3.35 visualizes an example audio input, an extracted example feature, and the mean and standard deviation extracted per texture window from the feature series.

3.7.3 Normalization and Mapping

The normalization of features is a trivial but potentially crucial step for many inference systems. As different features with different ranges and distributions are combined into observation vectors, proper feature scaling ensures that one feature does not have outsized impact on the training or the inference result. Consider two identical features with identical distributions except for a scaling factor λ. Each observation vector contains the two features. Although both features carry the same information beyond the scaling factor, they will impact a distance measure in this two-dimensional space differently. Computing the squared Euclidean distance, for instance, the second dimension's distance will have the weight λ^2 while the first dimension will be weighted with 1. Large λ will thus let the second feature dominate the distance while small λ will render the second dimension superfluous.

The goal of feature normalization is to minimize these scaling differences. A system should not, for example, give different results because the feature extractor returns the spectral centroid in Hertz instead of bins. A common approach to normalize features if they all have *symmetric* distributions with *identical shape* is to remove their mean value and scale them to a standard deviation of 1 (see, e.g. [81]):

$$v_{j,\mathrm{N}}(n) = \frac{v_j(n) - \mu_j}{\sigma_j}. \tag{3.71}$$

This normalization is referred to as *z-score normalization*. Note that the resulting normalized features are not bounded. Z-score normalization ensures similar impact of individual features as long as the features are distributed symmetrically and similar to a Gaussian distribution. However, if the distributions of different features are different and not Gaussian, then this normalization has to be applied with care.

A second common feature normalization is the so-called *min-max normalization* (also referred to as min–max scaling) which normalizes each feature to a range $[0, 1]$:

$$v_{j,\mathrm{N}}(n) = \frac{v_j(n) - \min(v_j)}{\max(v_j) - \min(v_j)}. \tag{3.72}$$

This normalization results in bounded and positive features, but infrequent outliers might impact the range.

Note that some literature will refer to these two standard approaches to feature scaling with different terminology: what is referred to here as z-score normalization is sometimes called standardization, and what is referred to here as min-max normalization might just be called normalization.

The normalization constants $\mu_{v_j}, \sigma_{v_j}, \max(v_j), \min(v_j)$ have to be estimated from the *training data*. The same (training) constants are then applied during inference. Extracting constants from the test data is incorrect as the system has to infer with exactly the same parameters as during training.

If the features have dramatically different and skewed distributions, the above normalizations might not be applicable. The normalization of slightly skewed features might improve when the median $Q_j(0.5)$ replaces μ_j (and in the computation of the standard deviation σ_j).

For more skewed features, the transformation of the feature to a target distribution (e.g. a Gaussian distribution) might help. Several approaches exist to transform a given distribution into a Gaussian distribution; widely known is the *Box–Cox transform* [82]. One example of this transform is

$$v^{(\beta)} = \begin{cases} \frac{v^{\beta}-1}{\beta}, & \beta \neq 0 \\ \log(v), & \beta = 0 \end{cases} \tag{3.73}$$

with the parameter β to be estimated. The Box–Cox transform only considers a limited class of transformation functions and thus does not guarantee that any arbitrary distribution can be mapped to a Gaussian distribution. There are also numerical methods of finding appropriate feature transformations. One example is the work of Albada and Robinson who transform arbitrary distributions to the normal distribution [83]. The transformation function for every feature has then to be stored numerically.

In many practical applications, it is sufficient to transform selected features in a way that results only in roughly approximated Gaussian distributions. Only features failing a test for Gaussianity have to be subjected to a transformation. Statistical procedures to test a distribution for Gaussianity are, for example, the Kolmogorov–Smirnov test, the Lilliefors test, the Shapiro–Wilk test [84], and the Anderson–Darling test [85]. Thode gives a good introductory overview [86]. Unfortunately, the result of these statistical tests is only of limited use in ACA because these tests nearly always tend to fail for large numbers of observations. In these cases, it is more practical to compute both the skewness and possibly the kurtosis (see Section 3.5.3) of the features to determine how *Gaussian* their distribution is. As a rule of thumb, distributions with a skewness between −2 and 2 are not significantly skewed and can thus be considered to be symmetric [87].

The normalization of multidimensional features requires special consideration and depends on both the feature characteristics as well as the specific use case. The following examples show the variety of approaches:

- MFCCs (Section 3.5.7): Each coefficient is usually considered independent and therefore normalized independently as if they were different features.
- Pitch Chroma (Section 7.5.1): Treating each pitch class independently would distort the pitch class relationships in the feature. The common choice is to normalize each observation (i.e. vector per block) independently and thus preserve pitch class relationships while sacrificing volume information over time. Thus, each 12-dimensional pitch chroma observation is usually either normalized to

a sum of 1 (like a pitch distribution) or to a vector length of 1 (like a point in Cartesian pitch space) [88].

- (Mel) Spectrogram (Section 3.3.3): a magnitude spectrogram is most commonly normalized to a range between 0 and 1 by dividing all values by its maximum value. There have been, however, also proposals to apply a localized normalization strategy [89].

3.7.4 Feature Dimensionality Reduction

Although large numbers of features can easily be extracted from the audio data, it is unclear a priori which features will help the final inference (in the following, we will only refer to classifiers as a placeholder for other inference stages) of the ACA system.

While it might seem intuitive to assume that any classifier will benefit from more information at the input (i.e. from a higher-input dimensionality of each observation), this is not necessarily true. In fact, increasing the input dimensionality can require a higher-model complexity, which in turn increases the likelihood of *overfitting*. This means that while the model fits the training data well, it does not generalize to unseen data [90] (see Section 4.5). A closely related problem is that with higher-input dimensionality, the sparseness of the feature space tends to increase, which in turn can make it harder to build a general machine learning model; this is sometimes referred to as the *curse of dimensionality* [91]. Figure 3.36 shows the output accuracy of a simple speech/music classifier over the number of features at the classifier input. While this task appears easy to solve with just a handful of features, it can be observed that after adding more and more features, classifier performance starts to drop.

This clearly indicates a need to limit the overall number of features. There might also be other reasons to make decreasing the number of features desirable, such as reduced computational complexity or increased interpretability.

Therefore, depending on the classifier, the task complexity, the amount of training data, and other requirements, it can be beneficial to reduce the feature

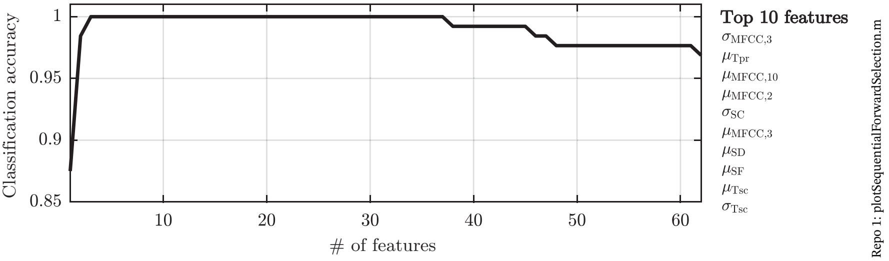


Figure 3.36 Accuracy over number of features selected by Sequential Forward Selection.

dimensionality. As some classifiers are more robust against high-dimensional input dimensions than others, no clear recommendations can be made regarding the optimal number of features; often, the number of features that maximizes validation accuracy is picked. In traditional music classification tasks, the feature vector dimensionality most commonly ranged from 20 to 200.

When reducing the dimensionality of the feature set, the goal is to remove redundant and irrelevant information while retaining the (task-) relevant information. Methods for feature dimensionality reduction can be grouped into two categories:

- *feature subset selection* to discard specific features, and
- *feature space transformation* to transform the features to a lower-dimensional space.

In the first case, the most promising feature subset is chosen; in the latter case, only those dimensions in the transformed feature space are discarded that contribute the least information.

Note that the requirement of feature dimensionality reduction strongly depends on the classification algorithm used; furthermore, it is still under discussion whether complex methods of dimensionality reduction really outperform simple ones [92].

3.7.4.1 Feature Subset Selection

The aim of *feature subset selection* is to reduce the number of used features by discarding the least powerful features. Formally, the available feature set

$$\mathcal{V} = v_j|_{j=1,\ldots,\mathcal{F}} \tag{3.74}$$

should be reduced to the feature subset

$$\mathcal{V}_\mathrm{s} = v_j|_{j=1,\ldots,\mathcal{F}_\mathrm{s}} \tag{3.75}$$

with $\mathcal{F}_\mathrm{s} < \mathcal{F}$; the chosen subset optimizes a given objective function $J(\mathcal{V}_\mathrm{s})$.

Feature selection algorithms are called *wrapper methods* if the objective function is the classifier itself and *filter methods* if the objective function $J(\mathcal{V}_\mathrm{s})$ is independent of the classification system used. Filter methods select features based on properties a good feature set is presumed to have and are usually less computationally expensive than wrapper methods.

This section will only provide a short introduction to the most common approaches to feature subset selection. More in-depth surveys of this topic have been published by Guyon and Elisseefi [93] and Cantú-Paz et al. [94].

Examples for wrapper methods are

- *Brute force subset selection*: The most obvious way of finding the optimal feature subset is to compute the classification accuracy for all possible combinations of features and to select the subset that performs best. The disadvantage of this approach is that the number of subsets to test, i.e. to train and evaluate, will be $2^{\mathcal{F}}$. This renders this method impractical for large numbers of features $\mathcal{F}$.

- *Single variable classification*: A simple feature ranking can be obtained by calculating the classification accuracy for each individual feature. Both features performing well and features performing poorly can thus be easily identified. While selecting the high-performing features and discarding the features that perform worst seems to be an intuitive solution, there are two potential issues with selecting or discarding features this way:
  1. The feature ranking contains no information on the *correlation* of two or more features. Consider the case of two features with identical values. They will both have the same ranking which is possibly high, but leaving both in the selected feature subset cannot improve classifier performance (and might even harm it in some cases).
  2. The feature ranking contains no information on the *combined usefulness* of features. A feature that adds no information individually might be able to add information in combination with other features.
- *Sequential forward selection*: *Sequential forward selection* starts with an empty subset of features. In the first iteration, it considers all feature subsets with only one feature (i.e. the *single variable classification* approach). The feature leading to the highest classification accuracy is used to initialize the set of selected features. From the remaining set of features, each iteration adds the feature that results in the highest increase of classification accuracy in combination with the set of previously selected features. This iterative algorithm can be summarized into the following processing steps:
  1. Start with an empty feature subset $\mathcal{V}_\mathrm{s} = \emptyset$.
  2. Find the one feature v_j not yet included in the feature subset that in combination with the subset maximizes the objective function:

$$v_j = \underset{\forall j | v_j \notin \mathcal{V}_\mathrm{s}}{\operatorname{argmax}} J(\mathcal{V}_\mathrm{s} \bigcup v_j). \tag{3.76}$$

  3. Add feature v_j to $\mathcal{V}_\mathrm{s}$.
  4. Go to step 2 and repeat the procedure until the required number of features has been selected or another termination criterion (e.g. decrease in validation set accuracy) has been triggered.

  Listing 3.20 shows example code for sequential forward selection. In this case, the accuracy per feature is estimated with a Nearest Neighbor classifier through Leave-One-Out Cross Validation (see below in Chapter 4). The accuracy over the number of selected features for a simple 2-class speech/music classification problem is shown in Figure 3.36.

  A closely related way of feature selection is *Sequential backward elimination*, which works in an analogous way to sequential forward selection but starts

```
    for f = 1:iNumFeatures
        % accuracy of selected features plus current feature f
        acc(f) = ToolLooCrossVal(V(f, :), ClassIdx);
    end
    [maxacc, maxidx] = max(acc);
    selFeatureIdx(1) = maxidx;
    unselectedFeatures(maxidx) = 0;
    AccPerSubset(1) = maxacc;

    % iterate until target number of features is reached
    for i = 2:iNumFeatures2Select
        acc = zeros(1, iNumFeatures);

        % iterate over all features not yet selected
        for f = 1:iNumFeatures
            if (unselectedFeatures(f) > 0)
                % accuracy of selected features plus current feature f
                acc(f) = ToolLooCrossVal(...
                    V([selFeatureIdx(1:(i-1)) f], :), ...
                    ClassIdx);
            else
                acc(f) = -1;
                continue;
            end
        end

        % identify feature maximizing the accuracy
        % move feature from unselected to selected
        [maxacc, maxidx] = max(acc);
        selFeatureIdx(i) = maxidx;
        unselectedFeatures(maxidx) = 0;
        AccPerSubset(i) = maxacc;
    end
```

Repo 2: ToolSeqFeatureSel.m
Repo 3: ToolSeqFeatureSel.py

Listing 3.20 Example code for Sequential Forward Feature Selection.

with a full subset of features and iteratively removes features from the subset. It is computationally less efficient than sequential forward selection, however, Sequential backward elimination can be argued to give better results since sequential forward selection does not assess the importance of features in combination with other not yet included features.

There exist several filter methods for finding variable rankings, including, for example, Chi-square statistics or any arbitrary separability measure.

3.7.4.2 Feature Space Transformation

The objective of *feature space transformation* is to reduce the number of used features by transforming them into a lower-dimensional space.

A frequently used feature space transformation is *Principal Component Analysis (PCA)*. PCA, summarized in Appendix C, does not reduce the dimensionality of the data by itself. However, it transforms the features into new dimensions which are sorted according to the variance they contribute to the data. This order is often interpreted as a measure of importance, so that the components that account for low variance can be discarded. A widely used systematic criterion to decide how many components can be discarded is based on the eigenvalue. This approach is based on the assumption that every component with an eigenvalue lower than 1 can be discarded. This criterion is equivalent of a threshold of $^1/_{\mathcal{F}}$ for the relative variance for which a component accounts. In many cases, this criterion leaves more components in the data set than useful. A slightly more “manual” approach is to visually identify either the index after which the eigenvalues are significantly lower or the index after which eigenvalues tend to be very similar to each other and remove components after that.

Other transformation methods can be used for feature space transformation but will not be explained in detail in this book. Typical approaches that can be found in the literature are *Independent Component Analysis (ICA)* and *Singular Value Decomposition (SVD)*. *Linear Discriminant Analysis (LDA)* also transforms the feature space, however, the number of output components cannot be chosen freely in this case. The main distinction of LDA and PCA is that PCA maximizes the variance and LDA maximizes class separability.

Using transformation methods for dimensionality reduction has two potential disadvantages, (i) the transformed features cannot be interpreted as easily as the original features since the transformed features are (linear) combinations of the original features, and (ii) unlike feature selection, the workload cannot be reduced since all input features have to be computed regardless of their importance.

3.8 Exercises

3.8.1 Questions

1. How are (fundamental) frequency f_0 and period length T_0 of a periodic signal related?
2. What are the frequencies of the first 5 harmonics of a periodic signal with a fundamental frequency $f_0 = 220$ Hz?
3. Is the signal $x(t) = A \cdot \cos(2\pi t n(t))$ (with A constant, $n(t)$ a Gaussian-distributed random variable, and t the time) a deterministic or a random signal?
4. Is it possible to compute the Fourier Series coefficients of a random white noise input signal?
5. Consider a periodic signal with period length 4. What is the 0th Fourier coefficient for this signal given that the area under the signal curve for one period is 8?

6. A continuous random variable $X \in [0, 2]$ has a uniform distribution. What is the mean of the random variable? What is the variance of this variable?
7. Sketch the Fourier Transform of two delta impulses located symmetrically around $t = 0$.
8. What happens to the Fourier Transform of a signal if it is multiplied (in the time-domain) with a cosine with nonzero frequency?
9. Given that the Fourier transform of $x(t)$ is $X(\mathrm{j}\omega)$, what is the Fourier transform of $x(t - T_1)$?
10. Discuss the differences between a Mel Spectrogram and a 'normal' spectrogram.
11. Given a signal $x(t) = A\cos(2\pi ft)$ ($A = 0.5, f = 440$ for $0 \leq t < 1$ s and $A = 0.9$, $f = 440$ for 1 s $\leq t < 2$ s). Which of the following features would be the best choice to detect the change in the amplitude?
    - Spectral Centroid
    - Spectral Flux
    - Spectral Spread
    - Spectral Skewness
12. Complete the sentence: The Spectral Centroid has been shown to correlate to some degree to the human perception of
    (i) roughness,
    (ii) brightness,
    (iii) happiness,
    (iv) loudness.
13. What is the Spectral Centroid of a zero input signal?
14. What is the Spectral Flux of an infinite sinusoidal? Assume the sine frequency to match a bin frequency of the spectrum.
15. One row of a feature matrix with mean $\mu = 5$ and standard deviation $\sigma = 2$ is passed through a z-score normalization block. What is the mean and standard deviation of the resulting row?
16. A feature matrix $\boldsymbol{V}$ of size $m \times n$ ($m = 5$ is the number of features per block and $n = 2000$ is the number of blocks of the input audio) is aggregated over all n. If we use mean, standard deviation, and range as the three aggregating metrics, what will be the dimension of the resulting aggregated feature matrix?
17. List advantages and potential drawbacks of feature learning as opposed to using traditional instantaneous features.
18. Summarize typical problems with high-dimensional data in machine learning.
19. We apply a PCA to a feature matrix $\boldsymbol{V}$ of size $m \times n$ ($m = 5$ is the number of features per block and $n = 2000$ is the number of blocks of the input audio) to get a transformed feature matrix $\boldsymbol{V}_{\mathrm{PCA}}$. Given the equation of PCA transformation is given as $\boldsymbol{V}_{\mathrm{PCA}} = \boldsymbol{T}^{\mathrm{T}}\boldsymbol{V}$, what is the dimension of the transformation matrix $\boldsymbol{T}$?

20. We apply a PCA to a feature matrix $\boldsymbol{V}$ of size $m \times n$ ($m = 5$ is the number of features per block and $n = 2000$ is the number of blocks of the input audio) to get a transformed feature matrix $\boldsymbol{V}_{\mathrm{PCA}}$. Given the equation of PCA transformation is given as $\boldsymbol{V}_{\mathrm{PCA}} = \boldsymbol{T}^{\mathrm{T}}\boldsymbol{V}$, what is the dimension of the transformation matrix $\boldsymbol{V}_{\mathrm{PCA}}$?

3.8.2 Assignments

1. Preprocessing
    - (a) Implement a function `[fs, x] = read_audio(cPath2Audio)` which reads a wave audio file with scipy.io.wavfile.read and returns the sample rate `fs` and a numpy array `x` with dimensions `numSamples` × `numChannels`. Ensure that the maximum possible value range of `x` is always $[-1, 1[$ regardless of the wordlength of the input file. Support the following wordlengths/input types: `uint8`, `int16`, `int32`, `float32`
    - (b) Implement a function `[x_m] = downmix_audio(x)` which takes a numpy array `x` with dimensions `numSamples` × `numChannels` and returns the downmixed numpy array `x_m` with dimensions `numSamples` × 1.
    - (c) Implement a function `[x_n] = normalize_audio(x_m)` which takes a numpy array `x_m+ with dimensions \verb+numSamples` × 1 and returns the normalized numpy array `x_n` of the same dimensions but scaled so the `max(abs(x_n)) = 1`.
2. Filtering
    - (a) Generate a uniformly distributed white noise signal of length three seconds at a sample rate of 48 kHz.
    - (b) Implement a function `[x_f] = filter_ma(x_m, order)` that filters a signal `x_m` with a MA filter of length `order`.
    - (c) Implement a function `[x_f] = filter_spa(x_m, alpha)` that filters a signal `x_m` with a single-pole filter with coefficient `alpha`.
    - (d) Compute the filtered noise signals for the following parametrizations
        - `order = 5`
        - `order = 10`
        - `order = 256`
        - `alpha = 0.01`
        - `alpha = 0.1`
        - `alpha = 0.5`
        - `alpha = 0.9`
        - `alpha = 0.99`

      and compute the magnitude spectra of the outputs. Compare the spectra with the magnitude spectrum of the input and with the other spectra. Listen to the filter noise. Describe what is happening.

3. Blocking
    (a) Implement a function `[x_B, t_B] = block_audio(x_m, blockSize, hopSize, fs)` which returns a numpy array `x_B` (dimension `N` × `blockSize`) and a vector t_B (dimension `N` × 1) for blocking the input audio signal into overlapping blocks. Make sure that each sample of the audio signal is included in at least one of the `N` blocks. `t_B` will refer to the start time of each block in seconds.
4. Spectrogram
    (a) Implement a function `[w] = compute_hann(windowLength)` that computes a Hann window of length `windowLength` and returns the window in numpy vector `w`. Before you implement, investigate the difference between choosing a "symmetric" vs. a "periodic" window function and choose the "periodic" implementation.
    (b) Implement a function `[X, fInHz, t_B] = compute_spectrogram(x_B, t_B, fs)` that computes the magnitude spectrum for each block of audio in `x_B` and returns the magnitude spectrogram `X` (dimensions `blockSize/2+1` × `N`), a frequency vector `fInHz` (dimensions `blockSize/2+1` × 1) containing the central frequency of each bin, and the previously computed time stamp vector `t_B`.
    For the computation of the spectrogram, use the Hann window computed with `compute_hann`. Scale your Fourier transform with `2/blockSize` to ensure the magnitude ranges don't vary between different window lengths. Note: remove the redundant part of the spectrum if necessary.
    (c) Visualize the spectrogram in Decibels for a selection of short audio files from your computer. Explore different 3D-plotting functions.
5. Feature Extraction, Aggregation, and Normalization
    (a) Implement the following five functions to extract audio features:
        - `[v_sc] = extract_spectral_centroid(x_B, fs)`
        - `[v_rms] = extract_rms(x_B)` (in dBFS)
        - `[v_zcr] = extract_zerocrossingrate(x_B)`
        - `[v_scr] = extract_spectral_crest(x_B)`
        - `[v_sf] = extract_spectral_flux(x_B)`

        with each function returning a numpy vector equaling the number of blocks in `x_B`. Use the function `compute_spectrogram` internally for the spectral features. Implement each of these functions in the default formulation from the text book/slides. Make sure all of your functions can properly handle zero input without return `nan`, `inf`, or throwing exceptions.
    (b) Implement a function: `[features] = extract_features(x_m, blockSize, hopSize, fs)`. This function serves as a wrapper function for your feature extraction functions. It will take a single-channel

audio vector `x` as input, blocks the audio with the specified block size and hop size using `block_audio`, calls each of the feature extractor functions implemented above and returns a numpy array of dimensions 5× `N` as feature matrix.

   (c) Implement a function: `[features_agg] = aggregate_feature_per_file(features)`. This function aggregates the feature matrix returned by `extract_features`, aggregates the features per audio file across blocks using mean and standard deviation and returns an aggregated feature matrix of dimensions 10×1 (first, the mean of all extracted features, then the standard deviation of all extracted features).
   (d) Implement a function: `[features_all] = get_feature_data (path2Audio, blockSize, hopSize)`. This function loops over all audio files contained within a folder indicated in `path2Audio`. The function will then successively call `extract_features` and `aggregate_feature_per_file`. The function will return `features_all`, a 10 × M numpy feature matrix which contains the aggregated features for the M audio files contained in the directory.
   (e) Implement a function: `[features_norm] = normalize_zscore (features_all)`. This function applies the z-score normalization method to the feature matrix `features_all`. The z-score normalization normalizes each feature to a zero mean and unit standard deviation across the entire dataset (of M files). The output array has the same dimensions as the input. Be careful to normalize the correct dimensions.
6. Feature Visualization
   (a) Download a small dataset, for example, a music/speech dataset[4]
   (b) Implement a function `visualize_features(path2Audio)` which first extracts two separate feature matrices for music and speech by using `get_feature_data` with `blockSize` 1024 and `hopSize` 256. Second, the two feature matrices will be normalized over the entire dataset. Note that this means you have to normalize both feature matrices with the same z-score parametrization. (hint: concatenate the two feature matrices appropriately before calling `normalize_zscore`). Third, write code that generates plots visualizing the feature space of the following five pairs of features:
      - `[SC mean, SCR mean]`,
      - `[SF mean, ZCR mean]`,
      - `[RMS mean, RMS std]`,
      - `[ZCR std, SCR std]`,
      - `[SC std, SF std]`.

---

[4] MusicSpeech. http://opihi.cs.uvic.ca/sound/music_speech.tar.gz. Last retrieved on Nov. 27, 2021

Note that you want to plot the data points corresponding to music with a different color than the points that correspond to speech, e.g. use red for music and blue for speech. This type of visualization is referred to as scatter plot. What can you infer from these scatter plots?

(c) Compute a feature covariance matrix. What can you see from the results? How are the features related to each other?

7. Feature Transformation
   - Taking the `features_norm` matrix for the dataset above, explore how many principal components you can find with PCA. Use the `ToolPca` function provided with the scripts accompanying this book. Note: be careful about input and output dimensionality.
   - Plot the Eigenvalues and discuss the dimensionality of the transformed feature space.
   - Assess the transformation matrix and see if you derive an interpretation of what each individual component means.

References

1 J Willard Gibbs. Fourier's Series. *Nature*, 59(1522):200–200, December 1898. ISSN 0028-0836. doi: 10.1038/059200b0. URL http://www.nature.com/doifinder/10.1038/059200b0.

2 J Willard Gibbs. Fourier's Series. *Nature*, 59(1539):606–606, April 1899. ISSN 0028-0836. doi: 10.1038/059606a0. URL http://www.nature.com/doifinder/10.1038/059606a0.

3 Julius O Smith and Phil Gosset. A Flexible Sampling-rate Conversion Method. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 2, San Diego, CA, 1984.

4 Judith C Brown. Calculation of a Constant Q Spectral Transform. *Journal of the Acoustical Society of America (JASA)*, 89(1):425–434, 1991. ISSN 00014966. doi: 10.1121/1.400476. URL http://link.aip.org/link/JASMAN/v89/i1/p425/s1&Agg=doi.

5 Judith C Brown and Miller S Puckette. An Efficient Algorithm for the Calculation of a Constant Q Transform. *Journal of the Acoustical Society of America (JASA)*, 92(5):2698, 1992. ISSN 00014966. doi: 10.1121/1.404385. URL http://link.aip.org/link/JASMAN/v92/i5/p2698/s1&Agg=doi.

6 Christian Schörkhuber and Anssi P Klapuri. Constant-Q Transform Toolbox for Music Processing. In *Proceedings of the 7th Sound and Music Computing Conference (SMC)*, Barcelona, 2010.

7 Richard F Lyon, Andreas G Katsiamis, and Emmanuel M Drakakis. History and future of auditory filter models. In *Proceedings of the International*

Symposium on Circuits and Systems (ISCAS), pages 3809–3812. IEEE, May 2010. ISBN 978-1-4244-5308-5. doi: 10.1109/ISCAS.2010.5537724. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5537724.

8 E de Boer and H R de Jongh. On Cochlear Encoding: Potentialities and Limitations of the Reverse-correlation Technique. *Journal of the Acoustical Society of America (JASA)*, 63(1):115, January 1978. ISSN 00014966. doi: 10.1121/1.381704. URL http://link.aip.org/link/JASMAN/v63/i1/p115/s1&Agg=doi.

9 A M H J Aertsen and P I M Johannesma. Spectro-Temporal Receptive Fields of Auditory Neurons in the Grassfrog. *Biological Cybernetics*, 38(4):223–234, November 1980. ISSN 0340-1200. doi: 10.1007/BF00337015. URL http://www.springerlink.com/index/10.1007/BF00337015.

10 Malcolm Slaney. An Efficient Implementation of the Patterson-Holdsworth Auditory Filter Bank. Technical report, Apple Perception Group, 1993.

11 Luc Van Immerseel and Stefaan Peeters. Digital Implementation of Linear Gammatone Filters: Comparison of Design Methods. *Acoustics Research Letters Online*, 4(3):59, 2003. ISSN 15297853. doi: 10.1121/1.1573131. URL http://link.aip.org/link/ARLOFJ/v4/i3/p59/s1&Agg=doi.

12 Alexander Lerch. Ein Ansatz zur automatischen Erkennung der Tonart in Musikdateien. In *Proceedings of the VDT International Audio Convention (23. Tonmeistertagung)*, Leipzig, November 2004. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2016/10/Lerch-2004-Ein-Ansatz-zur-automatischen-Erkennung-der-Tonart-.pdf.

13 Borching Su and Shyh-Kang Jeng. Multi-Timbre Chord Classification using Wavelet Transform and Self-Organized Map Neural Networks. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE, 2001. ISBN 0-7803-7041-4. doi: 10.1109/ICASSP.2001.940383. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=940383.

14 Eric D Scheirer. Towards Music Understanding without Separation: Segmenting Music with Correlogram Comodulation. In *Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 99–102, New Paltz, NY, USA, October 1999. doi: 10.1109/ASPAA.1999.810859.

15 Sebastian Wegener, Martin Haller, Juan José Burred, Thomas Sikora, Slim Essid, and Gaël Richard. On the Robustness of Audio Features for Musical Instrument Classification. In *Proceedings of the 16th European Signal Processing Conference (EUSIPCO)*, Lausanne, 2008.

16 ISO/IEC JTC1/SC29 15938-4:2002. Information Technology – Multimedia Content Description Interface – Part 4: Audio. Standard 15938-4:2002, ISO/IEC, 2002.

17 Geoffroy Peeters. A Large Set of Audio Features for Sound Description (Similarity and Classification) in the CUIDADO Project. Project Report (CUIDADO), IRCAM, 2004.

18 Rudolf A Rasch and Reinier Plomb. The Perception of Musical Tones. In Diana Deutsch, editor, *The Psychology of Music*. Academic Press, New York, 1982.

19 Eberhard Zwicker and Richard Feldtkeller. *Das Ohr als Nachrichtenempfänger*. S. Hirzel Verlag, Stuttgart, 2nd edition, 1967. ISBN 978-3-7776-0104-5.

20 Brian C J Moore. *An Introduction to the Psychology of Hearing*. Academic Press, London, 4th edition, 1997.

21 ASA. Acoustical Terminology. Standard, American Standards Association (ASA), 1960.

22 Albert S Bregman. *Auditory Scene Analysis*. MIT Press, 1994.

23 J Fred Blumenbach. *The Elements of Physiology*. Longman, Rees, Orme, Brown, and Green, London, 4th edition, 1828.

24 Hermann Von Helmholtz. *Die Lehre von den Tonempfindungen als physiologische Grundlage für die Theorie der Musik*. Vieweg, Braunschweig, 3rd edition, 1870.

25 Carl Stumpf. *Tonpsychologie II*. Knuf and Bonset, Hilversum / Amsterdam, 1890.

26 Carl E Seashore. *Psychology of Music*. McGraw-Hill, New York, 1938.

27 Christoph Reuter. *Der Einschwingvorgang nichtperkussiver Musikinstrumente*. Peter Lang, Frankfurt, 1995.

28 Gottfried von Bismarck. Sharpness as an Attribute of the Timbre of Steady Sounds. *Acustica*, 30:159–172, 1974.

29 Paul Iverson and Carol L Krumhansl. Isolating the Dynamic Attributes of Musical Timbre. *Journal of the Acoustical Society of America (JASA)*, 94(5):2595–2603, 1993.

30 Stephen McAdams, Suzanne Winsberg, Sophie Donnadieu, Geert De Soete, and Jochen Krimphoff. Perceptual Scaling of Synthesized Musical Timbres: Common Dimensions, Specificities, and Latent Subject Classes. *Psychological Research*, 58(3):177–192, 1995. ISSN 0340-0727, 1430-2772. doi: 10.1007/BF00419633. URL http://link.springer.com/article/10.1007/BF00419633.

31 Stephen Lakatos. A Common Perceptual Space for Harmonic and Percussive Timbres. *Perception & Psychophysics*, 62(7):1426–1439, 2000.

32 Jeremy Marozeau, Alain De Cheveigné, Stephen McAdams, and Suzanne Winsberg. The Dependency of Timbre on Fundamental Frequency. *Journal of the Acoustical Society of America (JASA)*, 114(5):2946–2957, 2003.

33 Emery Schubert, Joe Wolfe, and Alex Tarnopolsky. Spectral Centroid and Timbre in Complex, Multiple Instrumental Textures. In *Proceedings of the 8th International Conference on Music Perception & Cognition (ICMPC)*, Evanston, August 2004.

34 Anne Caclin, Stephen McAdams, Bennett K Smith, and Suzanne Winsberg. Acoustic Correlates of Timbre Space Dimensions: A Confirmatory Study Using Synthetic Tones. *Journal of the Acoustical Society of America (JASA)*, 118(1):471–482, 2005. ISSN 00014966. doi: 10.1121/1.1929229. URL http://link.aip.org/link/JASMAN/v118/i1/p471/s1&Agg=doi.

35 Eberhard Zwicker and Hugo Fastl. *Psychoacoustics. Facts and Models*. Springer, 2nd edition, 1999. ISBN 978-3-540-23159-2.

36 Paul Masri and Andrew Bateman. Improved Modelling of Attack Transients in Music Analysis-Resynthesis. In *Proceedings of the International Computer Music Conference (ICMC)*, Hong Kong, 1996.

37 Arthur E Barnes. Instantaneous Spectral Bandwidth and Dominant Frequency with Applications to Seismic Reflection Data. *GEOPHYSICS*, 58(3):419–428, March 1993. ISSN 0016-8033. doi: 10.1190/1.1443425. URL https://library.seg.org/doi/abs/10.1190/1.1443425. Publisher: Society of Exploration Geophysicists.

38 Steven B Davis and Paul Mermelstein. Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences. *Transactions on Acoustics, Speech, and Signal Processing*, 28(4):357–366, August 1980. ISSN 0096-3518. doi: 10.1109/TASSP.1980.1163420. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1163420.

39 Jonathan T Foote. A Similarity Measure for Automatic Audio Classification. In *Proceedings of the 14th National Conference on Artificial Intelligence*, Providence, RI, 1997.

40 Beth Logan. Mel Frequency Cepstral Coefficients for Music Modeling. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Plymouth, 2000.

41 Elias Pampalk, Simon Dixon, and Gerhard Widmer. On the Evaluation of Perceptual Similarity Measures for Music. In *Proceedings of the 6th International Conference on Digital Audio Effects (DAFX)*, London, September 2003.

42 Juan José Burred and Alexander Lerch. A Hierarchical Approach to Automatic Musical Genre Classification. In *Proceedings of the 6th International Conference on Digital Audio Effects (DAFX)*, London, September 2003. doi: 10.1.1.2.6582. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2016/10/Burred-and-Lerch-2003-A-Hierarchical-Approach-to-Automatic-Musical-Genre.pdf.

43 George Tzanetakis and Perry Cook. Musical Genre Classification of Audio Signals. *Transactions on Speech and Audio Processing*, 10(5):293–302, July 2002. ISSN 1063-6676. doi: 10.1109/TSA.2002.800560. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1021072.

44 Martin F McKinney and Jeroen Breebart. Features for Audio and Music Classification. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Baltimore, MD, 2003.

45 Steve Young, Gunnar Evermann, Thomas Hain, Dan Kershaw, Gareth Moore, Julian Odell, Dave Ollason, Dan Povey, Valtcho Valtchev, and Phil Woodland. The HTK Book. Technical Report July 2000, 2002.

46 Malcolm Slaney. Auditory Toolbox – Version 2. Technical Report TR-1998-010, Interval Research Corporation, Wien, 1998.

47 Sirko Molau, Michael Pitz, Ralf Schlüter, and Hermann Ney. Computing Mel-frequency Cepstral Coefficients on the Power Spectrum. *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing. (ICASSP01)*, pages 73–76, 2001. doi: 10.1109/ICASSP.2001.940770. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=940770.

48 Nuggehally S Jayant and Peter Noll. *Digital Coding of Waveforms – Principles and Applications to Speech and Video*. Prentice Hall, New Jersey, 1984.

49 Thomas W Parsons. Separation of Speech from Interfering Speech by Means of Harmonic Selection. *Journal of the Acoustical Society of America (JASA)*, 60(4):911, 1976. ISSN 00014966. doi: 10.1121/1.381172. URL http://link.aip.org/link/JASMAN/v60/i4/p911/s1&Agg=doi. 00257.

50 Ernst Terhardt, Gerhard Stoll, and Manfred Seewann. Algorithm for Extraction of Pitch and Pitch Salience from Complex Tonal Signals. *Journal of the Acoustical Society of America (JASA)*, 71(3):679, 1982. ISSN 00014966. doi: 10.1121/1.387544. URL http://link.aip.org/link/JASMAN/v71/i3/p679/s1&Agg=doi.

51 Xavier Serra. *A System for Sound Analysis / Transformation / Synthesis Based on a Deterministic plus Stochastic Decomposition*. PhD thesis Dissertation, Stanford University, 1989.

52 Geoffroy Peeters and Xavier Rodet. Signal Characterization in terms of Sinusoidal and Non-Sinusoidal Components. In *Proceedings of the 1st Conference on Digital Audio Effects (DAFX)*, Barcelona, 1998.

53 Mathieu Lagrange, Sylvain Marchand, and Jean-Bernard Rault. Sinusoidal Parameter Extraction and Component Selection in a Non Stationary Model. In *Proceedings of the 5th International Conference on Digital Audio Effects (DAFX)*, Hamburg, 2002.

54 Mark Robert Every. *Separation of Musical Sources and Structure from Single-channel Polyphonic Recordings*. PhD thesis Dissertation, University of York, 2006.

55 Axel Röbel, Miroslav Zivanovic, and Xavier Rodet. Signal decomposition by means of classification of spectral peaks. In *Proceedings of the International Computer Music Conference (ICMC)*, Miami, 2004. ICMA.

56 Maciej Kulesza and Andrzej Czyzewski. Frequency Based Criterion for Distinguishing Tonal and Noisy Spectral Components. *International Journal of Computer Science and Security*, 4(1):1, March 2010. URL http://www.cscjournals.org/csc/manuscriptinfo.php?ManuscriptCode=77.74.67.68.39.47.48.104.

57 Sebastian Kraft, Alexander Lerch, and Udo Zölzer. The Tonalness Spectrum: Feature-Based Estimation of Tonal Components. In *Proceedings of the 16th International Conference on Digital Audio Effects*, Maynooth, 2013. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2015/04/Kraft%20et%20al_2013_The%20Tonalness%20Spectrum.pdf.

58 Eric J Humphrey, Juan P Bello, and Yann LeCun. Feature Learning and Deep Architectures: new Directions for Music Informatics. *Journal of Intelligent Information Systems*, 41(3):461–481, December 2013. ISSN 0925-9902, 1573-7675. doi: 10.1007/s10844-013-0248-5. URL http://link.springer.com/10.1007/s10844-013-0248-5.

59 Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. What's Hidden in a Randomly Weighted Neural Network? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11893–11902, Online, 2020. URL https://openaccess.thecvf.com/content_CVPR_2020/html/Ramanujan_Whats_Hidden_in_a_Randomly_Weighted_Neural_Network_CVPR_2020_paper.html.

60 Roger Grosse, Rajat Raina, Helen Kwong, and Andrew Y Ng. Shift-Invariant Sparse Coding for Audio Classification. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, UAI'07, pages 149–158, Arlington, Virginia, USA, July 2007. AUAI Press. ISBN 978-0-9749039-3-4.

61 Mikael Henaff, Kevin Jarrett, Koray Kavukcuoglu, and Yann LeCun. Unsupervised Learning of Sparse Features for Scalable Audio Classification. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 6, Miami, FL, 2011.

62 Konstantin Markov and Tomoko Matsui. Music Genre Classification Using Self-Taught Learning via Sparse Coding. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1929–1932, Kyoto, March 2012. Institute of Electrical and Electronics Engineers (IEEE). doi: 10.1109/ICASSP.2012.6288282. ISSN: 2379-190X.

63 Klaus Seyerlehner, Gerhard Widmer, and Peter Knees. Frame Level Audio Similarity - A Codebook Approach. In *Proceedings of the International Conference on Digital Audio Effects (DAFX)*, page 8, Espoo, Finland, 2008.

64 Jan Wülfing and Martin Riedmiller. Unsupervised Learning of Local Features for Music Classification. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Porto, Portugal, 2012.

65 Juhan Nam, Jorge Herrera, Malcolm Slaney, and Julius Smith. Learning Sparse Feature Representations For Music Annotation and Retrieval. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 6, Porto, Portugal, 2012.

66 Maria Panteli, Emmanouil Benetos, and Simon Dixon. Learning a Feature Space for Similarity in World Music. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 7, New York, 2016.

67 Siddharth Sigtia and Simon Dixon. Improved Music Feature Learning with Deep Neural Networks. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6959–6963, Florence, Italy, May 2014. Institute of Electrical and Electronics Engineers (IEEE). ISBN 978-1-4799-2893-4. doi: 10.1109/ICASSP.2014.6854949. URL http://ieeexplore.ieee.org/document/6854949/.

68 Honglak Lee, Peter Pham, Yan Largman, and Andrew Ng. Unsupervised Feature Learning for Audio Classification using Convolutional Deep Belief Networks. In *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*, volume 22, pages 1096–1104, 2009. URL https://proceedings.neurips.cc/paper/2009/hash/a113c1ecd3cace2237256f4c712f61b5-Abstract.html.

69 Philippe Hamel and Douglas Eck. Learning Features form Music Audio with Deep Belief Networks. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 6, Utrecht, Netherlands, 2010.

70 Sander Dieleman, Philemon Brakel, and Benjamin Schrauwen. Audio-Based Music Classification with a Pretrained Convolutional Network. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 6, Miami, FL, 2011.

71 Chih-Wei Wu and Alexander Lerch. Assessment of Percussive Music Performances with Feature Learning. *International Journal of Semantic Computing*, 12(3):315–333, 2018. ISSN 1793-351X. doi: 10.1142/S1793351X18400147. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2018/09/ws-ijsc_cw_submission.pdf.

72 Yoshua Bengio. Deep Learning of Representations for Unsupervised and Transfer Learning. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pages 17–36. JMLR Workshop and Conference Proceedings, June 2012. URL http://proceedings.mlr.press/v27/bengio12a.html. ISSN: 1938-7228.

73 Keunwoo Choi, György Fazekas, Mark B Sandler, and Kyunghyun Cho. Transfer Learning for Music Classification and Regression Tasks. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 141–149, Suzhou, China, 2017. URL https://ismir2017.smcnus.org/wp-content/uploads/2017/10/12_Paper.pdf.

74 Shawn Hershey, Sourish Chaudhuri, Dan P W Ellis, Jort F Gemmeke, Aren Jansen, R Channing Moore, Manoj Plakal, Devin Platt, Riff A Saurous, Bryan Seybold, Malcolm Slaney, Ron J Weiss, and Kevin Wilson. CNN Architectures for Large-Scale Audio Classification. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 131–135, New Orleans, LA, March 2017. Institute of Electrical and Electronics Engineers (IEEE). doi: 10.1109/ICASSP.2017.7952132. ISSN: 2379-190X.

75 Jason Cramer, Ho-Hsiang Wu, Justin Salamon, and Juan Pablo Bello. Look, Listen, and Learn More: Design Choices for Deep Audio Embeddings. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3852–3856, Brighton, UK, May 2019. Institute of Electrical and Electronics Engineers (IEEE). ISBN 978-1-4799-8131-1. doi: 10.1109/ICASSP.2019.8682475. URL https://ieeexplore.ieee.org/document/8682475/.

76 Jordi Pons and Xavier Serra. musicnn: Pre-Trained Convolutional Neural Networks for Music Audio Tagging. In *Late Breaking Demo (Extended Abstract), Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 2, Delft, Netherlands, 2019. URL http://arxiv.org/abs/1909.06654.

77 Mark Cartwright, Ana Elisa Mendez Mendez, Jason Cramer, Vincent Lostanlen, Graham Dove, Ho-Hsiang Wu, Justin Salamon, Oded Nov, and Juan Bello. SONYC Urban Sound Tagging (SONYC-UST): A Multilabel Dataset from an Urban Acoustic Sensor Network. In *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2019 Workshop (DCASE)*, October 2019. doi: 10.33682/j5zw-2t88. URL http://archive.nyu.edu/handle/2451/60776. Accepted: 2019-10-24T01:50:25Z.

78 Siddharth Gururani, Mohit Sharma, and Alexander Lerch. An Attention Mechanism for Music Instrument Recognition. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Delft, 2019. International Society for Music Information Retrieval (ISMIR). URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2019/07/Gururani-et-al.-2019-An-Attention-Mechanism-for-Music-Instrument-Recogn.pdf.

79 Scott Krig. *Computer Vision Metrics*. Springer International Publishing, Cham, 2016. ISBN 978-3-319-33761-6 978-3-319-33762-3. doi: 10.1007/978-3-319-33762-3. URL http://link.springer.com/10.1007/978-3-319-33762-3.

80 Fabian Mörchen, Alefred Ultsch, Michael Thies, and Ingo Löhken. Modeling Timbre Distance with Temporal Statistics from Polyphonic Music. *Transactions on Audio, Speech and Language Processing*, 14(1):81–90, January 2006. ISSN 1558-7916. doi: 10.1109/TSA.2005.860352. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1561266.

81 Selim Aksoy and Robert M Haralick. Feature Normalization and Likelihood-Based Similarity Measures for Image Retrieval. *Pattern Recognition Letters*, 22(5):563–582, 2001. ISSN 01678655. doi: 10.1016/S0167-8655(00)00112-4. URL http://www.sciencedirect.com/science/article/pii/S0167865500001124.

82 George E P Box and David R Cox. An Analysis of Transformations. *Journal of the Royal Statistical Society*, 26(2):211–252, 1964.

83 Sacha J Albada and Peter A Robinson. Transformation of Arbitrary Distributions to the Normal Distribution with Applications to EEG Test-retest Reliability. *Journal of Neuroscience Methods*, 161(2):205–211, 2007.

84 S S Shapiro and M B Wilk. An Analysis of Variance Test for Normality (Complete Samples). *Biometrica*, 52(3/4):591–611, 1965.

85 Theodore W Anderson and Donald A Darling. Asymptotic Theory of Certain "Goodness of Fit" Criteria Based on Stochastic Processes. *Annals of Mathematical Statistics*, 23(2):193–212, 1952.

86 Henry C Thode. *Testing for Normality*. Marcel Dekker, Inc., New York, 2002. ISBN 0-8247-9613-6.

87 Jeremy Miles and Mark Shevlin. *Applying Regression & Correlation: a Guide for Students and Researchers*. Sage Publications, London, 2001. ISBN 0-7619-6229-8.

88 Sebastian Ewert and Meinard Müller. Chroma Toolbox: Matlab Implementations for Extracting Variants of Chroma-Based Audio Features. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 6, Miami, FL, 2011.

89 Vincent Lostanlen, Justin Salamon, Mark Cartwright, Brian McFee, Andrew Farnsworth, Steve Kelling, and Juan Pablo Bello. Per-Channel Energy Normalization: Why and How. *IEEE Signal Processing Letters*, 26(1):39–43, January 2019. ISSN 1558-2361. doi: 10.1109/LSP.2018.2878620.

90 Douglas M Hawkins. The Problem of Overfitting. *Journal of Chemical Information and Computer Sciences*, 44(1):1–12, January 2004. ISSN 0095-2338. doi: 10.1021/ci0342472. URL https://pubs.acs.org/doi/10.1021/ci0342472.

91 Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 6th edition, 1972. ISBN 978-0-691-07951-6.

92 Rebecca Fiebrink and Ichiro Fujinaga. Feature Selection Pitfalls and Music Classification. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Victoria, 2006.

93 Isabelle Guyon and Andre Elisseefi. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, 3(7-8):1157–1182, October 2003. ISSN 1532-4435. doi: 10.1162/153244303322753616. URL http://www.crossref.org/jmlr_DOI.html.

94 Erick Cantú-Paz, Shawn Newsam, and Chandrika Kamath. Feature Selection in Scientific Applications. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, Seattle, August 2004. ACM Press. ISBN 1581138889. doi: 10.1145/1014052.1016915. URL http://portal.acm.org/citation.cfm?doid=1014052.1016915.

4

Inference

The second stage of the audio content analysis (ACA) system, the *inference*, takes the extracted features and maps them into a domain both usable and comprehensible by humans. In other words, it interprets the feature data and maps it to meaningful metadata, such as a class label for the example in Figure 3.21 (speech, chamber music, and pop) or the pitch of a melody. This inference system can be an expert-defined, rule-based algorithm, or a data-driven system like a classifier or a regression approach. The more condensed and the more meaningful the features are, the less powerful the inference algorithm has to be and vice versa: a raw feature representation close to the audio sample requires a sophisticated inference approach.

Historically, rule-based expert systems used to be common in audio analysis. Especially for simple tasks, using knowledge to implement a set of algorithmic rules solving the specific problem can be a viable solution. Part of the algorithms presented in this book will follow such an approach. For the majority of nontrivial (analysis) tasks, however, data-driven Machine Learning (ML) systems have shown superior performance. Machine-learning systems are based on more or less generic algorithms using data to adapt to specific use cases by parametrizing its algorithm adaptively. In order to do this properly, sufficient training data are required (see Section 5).

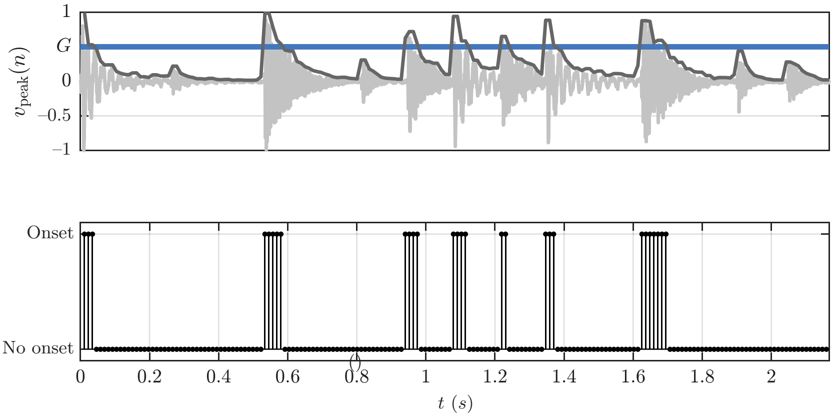
This chapter is not intended as a formal introduction into ML and classifiers. There exists a lot of introductory and advanced literature on the topic; see, for example, Bishop [1], Shalev-Shwartz and Ben-David [2], or Duda et al. [3]. Helpful books focusing on neural networks and deep learning have been written by Goodfellow et al. [4] and Ng [5].

An Introduction to Audio Content Analysis: Music Information Retrieval Tasks & Applications,
Second Edition. Alexander Lerch.
© 2023 The Institute of Electrical and Electronics Engineers, Inc. Published 2023 by John Wiley & Sons, Inc.

4.1 Classification

The most intuitive classifier is simply the application of a threshold: if a feature value is higher than a threshold G, choose class 1; otherwise, class 0. Figure 4.1 gives an example for this: the example feature is the peak envelope extracted from a drum loop; if the feature is higher than the threshold G, this block is classified as class 1 (onset); otherwise, as class 0 (no onset). The value of the threshold determines the ratio between class 1 and 0 detections. If the threshold is increased, fewer onsets will be detected and the likelihood of missing an onset rises, but at the same time, it is more certain that the detected onsets are correct detections. If the threshold is lowered, more onsets will be detected, increasing the likelihood of detecting all onsets but at the same time, it is more likely to incorrectly detect onsets (compare precision and recall metrics in Section 6).[1]

A simple data-driven system derives the threshold G from the data itself. A more sophisticated system might utilize multiple features and generalize to a multi-dimensional space with nonlinear thresholds. In other words, it learns what combination of feature values are common for each class and how to



Repo 1: plotThresholdClassification.m

Figure 4.1 Example for classifying the blocks of a drum loop into the classes onset and no onset based on the peak envelope of the signal and a simple threshold.

---

[1] Note that this example is not a recommended approach to onset detection. It can be seen in the visualization that even for a simple input signal such as a drum loop, several onsets have not been detected. Simple steps to improve this system could include (i) utilization of a more powerful feature than the envelope, (ii) computing the slope of the feature rather than the absolute value to analyze a measure of change, (iii) using an adaptive thresholding function, and (iv) postprocessing the results to combine nearby onsets into one onset. For more details on approaches to onset detection please refer to Section 9.3.

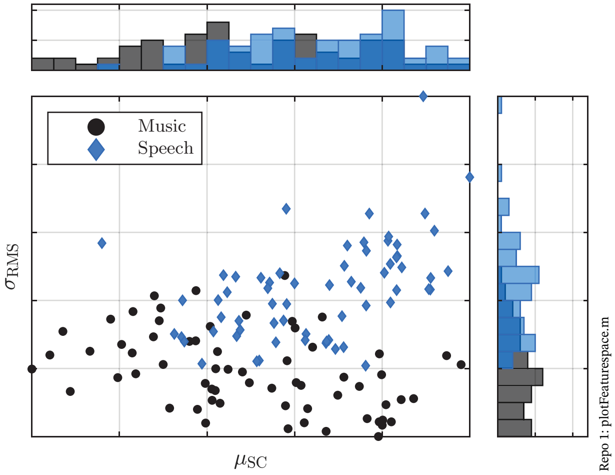


Figure 4.2 A music/speech dataset visualized in a two-dimensional feature space (x-axis: average spectral centroid, y-axis: standard deviation of RMS) and the feature distributions per class.

differentiate between classes given these feature values. Figure 4.2 shows a so-called “scatter plot” for two classes (speech and music) represented by two features (two-dimensional feature space). It can be seen that for these data, neither feature is able to separate the two classes well, although the RMS feature seems to work slightly better as the distribution of feature values has less overlap between classes. The fact that the classifier has to build the model from data emphasizes the importance of the so-called “training data set”; if the class distributions used to parametrize the classifier do not match the data sent into the classifier when testing, the classifier will not perform well.

A common classifier is the *Nearest Neighbor* classifier [6]. While training, it stores the location of each data point in the feature space with its corresponding class label. When queried with a new and unseen feature vector, the distance to every single previously stored training vector is computed; the final result is the class label of the closest vector, the *nearest neighbor*. An often-used variant is the *K-Nearest Neighbor (K-NN)* classifier, which uses not only the nearest neighbor as reference but also the nearest K training data points. As the estimated class label is then the majority of the labels of the K nearest vectors, K is usually an odd number (e.g. 3 or 7) in the case of two classes to have a clear majority. Figure 4.3 visualizes how the classification result might change for different K. In the case of more than two classes, it might occur that there is no clear class majority. This case can be handled by implementing fallback solutions such as taking into account the average distance per class or reducing K iteratively until a majority is found.

As the K-NN classifier is based on distance calculations, most commonly the Euclidean distance, feature normalization can have a major impact on

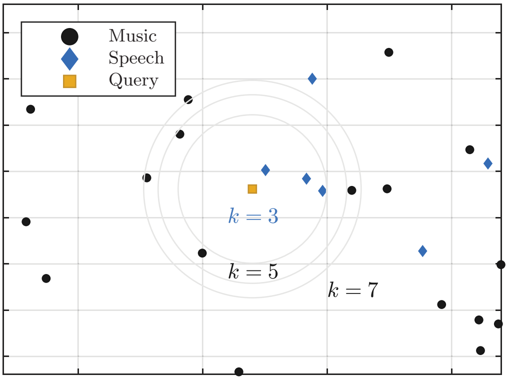


Figure 4.3 Nearest neighbor classification of a query data point (center) for different K. In this example, the result is the same for $K = 5$ and $K = 7$, but different for $K = 3$.

Repo 1: plotKNN.m

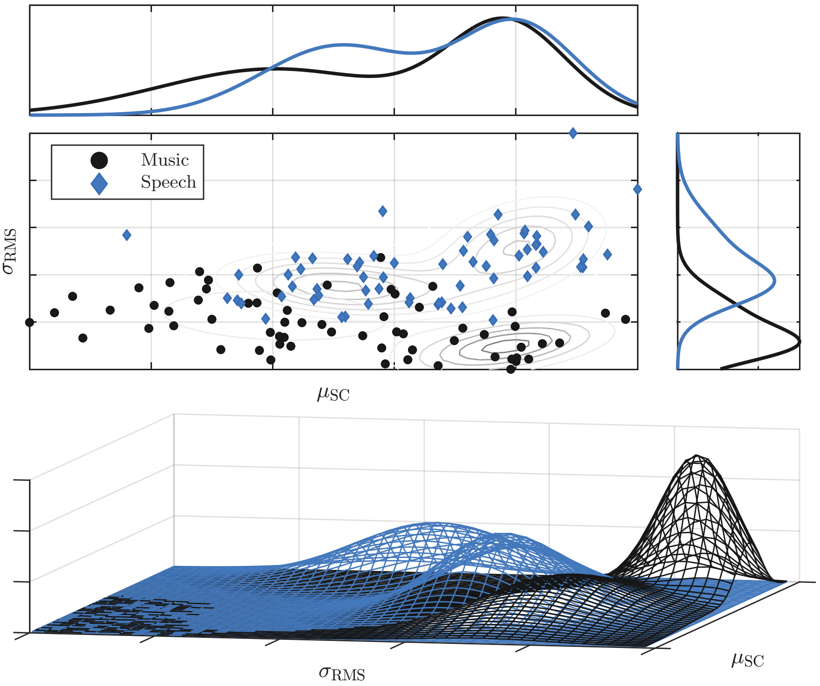
the classifier performance (see Section 3.7.3). Its simple implementation with few tunable parameters is an advantage. A feature vector with a too high dimensionality can affect classifier performance negatively; in this case, feature dimensionality reduction methods might help (see Section 3.7.4). Any Nearest Neighbor classifier learns by memorization, which means that the amount of data that needs to be stored for the model can be considerable as it includes all training observations. The distance computation to each stored data point can also make it expensive to execute. This makes K-NN a less-practical approach for large datasets. Listing 4.1 shows a simple implementation of a K-NN classifier.

```
% compute distances to all training observations
d = computeEucDist_I(TestFeatureVector, TrainFeatureMatrix);

% sort the distances to find closest
[dummy,idx] = sort(d);

% pick the majority of the k closest training observations
% note that for multi-class problems and even k, this needs to be
% refined
class = mode(TrainClassIndices(idx(1:k)));
```

Repo 2: ToolSimpleKnn.m
Repo 3: ToolSimpleKnn.py

Listing 4.1 Implementation of a K-NN classifier.

The vast majority of classifiers do not store all individual training data points but build a model from the data. Classification with Gaussian Mixture Models (GMMs), for example, assumes that the data distribution per class can be sufficiently modeled with multiple superposed Gaussian distributions. Figure 4.4 shows the two mixtures of two Gaussian distributions per class for the data example presented above. The distributions have been fitted with the Expectation Maximization (EM) algorithm [3]. In this classification scenario, each test observation is assigned the label of the mixture model with the higher value



Repo 1: plotGmm.m

Figure 4.4 The feature space from Figure 4.2 and a corresponding Gaussian Mixture Model with two Gaussians per class.

at this location. The value can also be interpreted as the likelihood of this test observation being this class. Listing 4.2 shows the implementation of a Gaussian Mixture Model (GMM).

The *Support Vector Machine (SVM)* is a classifier that has shown good performance for a large variety of tasks and is often the classifier of choice if deep learning is not suitable. It projects the data points into a higher-dimensional space and estimates a separating hyperplane that balances the margin between the classes (as large as possible) with the classification accuracy on the training data (as high as possible) [7]. Support Vector Machines SVMs also allow for complex classification scenarios by applying a nonlinear kernel to the data before estimating the separating hyperplane. This allows to go beyond linear classification even in the high-dimensional projected space.

The *Random Forest* classifier utilizes concepts from ensemble learning [8]. It is referred to as *forest* because it utilizes a (large) number of Decision Tree classifiers, and it is called “random” as it is trained with randomly selected subsets of both data and features. A *Decision Tree* is a classifier doing nested binary classifications to form nodes containing a large number of data points [9]. Different trees will

```
    for j = 1:numMaxIter
        prevState = state;

        % compute weighted gaussian
        p = computeProb_I(V, state);

        % update clusters
        state = updateGaussians_I(V, p, state);

        % if we have converged, break
        if (max(sum(abs(state.m-prevState.m))) <= 1e-20)
            break;
        end
    end

    mu = state.m;
    sigma = state.sigma;
end

function [state] = updateGaussians_I(FeatureMatrix, p, state)

    % number of clusters
    K = size(state.m, 2);

    % update priors
    state.prior = mean(p, 1)';

    for k = 1:K
        s = 0;

        % update means
        state.m(:, k) = FeatureMatrix * p(:, k) / sum(p(:, k));

        % subtract mean
        F = FeatureMatrix - repmat(state.m(:, k), 1, size(FeatureMatrix, 2));

        for n = 1:size(FeatureMatrix, 2)
            s = s + p(n, k) * (F(:, n) * F(:, n)');
        end
        state.sigma(:, :, k) = s / sum(p(:, k));
    end
end
```

Repo 2: ToolGmm.m
Repo 3: ToolGmm.py

Listing 4.2 Implementation of a GMM model.

be differently shaped as the derived decision thresholds depend on the provided training data and on the provided subset of features. Thus, they might or might not lead to the same classification result. The Random Forest aggregates all individual tree results and returns the majority vote. Generally, such ensemble methods tend to be robust and generalize well.

The *Hidden Markov Model (HMM)*, although commonly integrating a GMM, deserves a special mention since it does not only classify single instances individually but also models the probability of a sequence of observations. This is most frequently accomplished with the Viterbi algorithm (see Section 7.6) [10].

Deep Neural Networks (DNNs) have shown superior performance in nearly all Audio Content Analysis (ACA) tasks they have been applied to during the past decade. Therefore, the vast majority of state-of-the-art systems in the field of audio analysis are nowadays deep networks. Neural networks come in a large variety of different architectures [4]. As of writing this text, a very common classification setup for audio classification consists of a Convolutional Neural Network (CNN) with a Log-Mel Spectrogram input. The input representation preserves the majority of the relevant information of the audio signal, while the convolutional architecture allows for efficient internal data aggregation with a degree of translation invariance. Other architectures or architectural modifications such as recurrent neural networks or attention-based models have also been used successfully in audio analysis.

DNNs, however, are not necessarily always the best choice for all tasks despite their superiority in many scenarios. Many networks have a high complexity with a large number of hyperparameters; this requires (i) large amounts of training data that might not be easily available, (ii) potentially significant computational resources for training (and possibly inference), (iii) rigorous experimental setups to ensure proper hyperparameter tuning, and it also increases (iv) the likelihood of overfitting (see below) and other training issues. An important distinction between "traditional" classifiers and DNNs is that in the latter, the feature extraction and classification stages are not easily separated – while traditional classifiers mostly benefit from a powerful feature representation at their input, most DNN models are supposed to learn the relevant features from a raw (or less reduced) representation of the data. This removes the feature design and engineering from the list of tasks; this reduces the required expert knowledge utilized in the feature design and it allows to learn features from the data that can capture information potentially not covered by the custom-designed features (see Section 3.6).

4.2 Regression

Regression analysis also models the (statistical) relationship between a dependent variable (output) and one or more independent variables (features). Similar to classifiers, this model maps the input to the output. The difference to classifiers is that the prediction is a value as opposed to a categorical class label. Thus, regression models are commonly used for tasks that require the estimation of a continuous value given a set of features. That makes regression analysis a good fit for, for example, a standard mood recognition task which often aims at estimating the level of arousal and valence on two scales (see Section 13). Another example could be the grading of student music performances (see Section 15.3).

Linear Regression is a simple approach to regression that assumes a linear relationship between the input features and the output. In its simplest form, both the input and the predicted output values are one-dimensional (not vectors). Given a set of $\mathcal{N}$ feature observations (also: independent variables) v and corresponding target values (also: dependent variables) y, the model is trained by estimating the slope m and offset (also: bias, intercept) c of a straight line

$$\hat{y}(n) = m \cdot v(n) + c \tag{4.1}$$

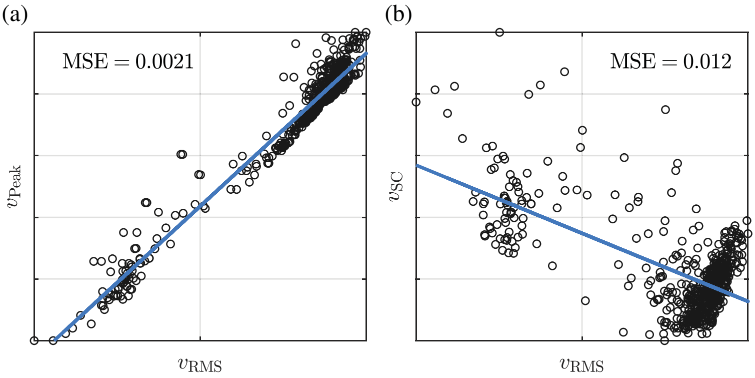
that minimizes the error between the model and the target given the features. The most common objective function to be minimized is the Mean Squared Error (MSE) (see Eq. (6.7)). With the goal of minimizing the MSE between the model and the targets, the following solution can be derived:

$$c = \mu_y - m \cdot \mu_v, \tag{4.2}$$

$$m = \frac{\sum\limits_{n=0}^{\mathcal{N}-1} \big(y(n) - \mu_y\big) \cdot \big(v(n) - \mu_v\big)}{\sum\limits_{n=0}^{\mathcal{N}-1} \big(v(n) - \mu_v\big)^2}. \tag{4.3}$$

The detailed derivation of these results can be found in Appendix D.

Figure 4.5 shows two example results; a linear model with a good fit between input and target (here: RMS and peak envelope, respectively) on the left (a) and a linear model with a bad fit between feature and target (here: RMS and spectral centroid, respectively) on the right (b).



Repo 1: plotLinearRegression.m

Figure 4.5 Linear Regression of two feature/target pairs, (a) RMS and peak envelope, (b) RMS and spectral centroid.

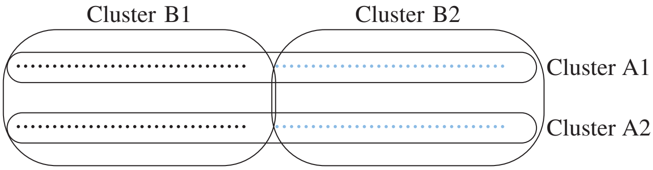


Figure 4.6 Example illustrating the importance of the similarity definition for clustering: the observations can be intuitively grouped according to spatial separation (clusters A) or according to color (clusters B).

In the case of multiple independent variables v, the model takes the form

$$\hat{y}(n) = m_1 v_1(n) + m_2 v_2(n) + \cdots + m_f v_f(n) + c. \tag{4.4}$$

There exist many other regression approaches, and most of them are more sophisticated and powerful than the simple linear regression introduced above. Two examples for nonlinear regression are Support Vector Regression (SVR) (compare SVM) and a DNN with a sigmoid output layer.

4.3 Clustering

Algorithms for *clustering* are usually unsupervised and are especially useful for exploratory analysis. The goal of clustering is to group observations in a way that similar observations are grouped together, while dissimilar observations end up in separate groups [2]. Obviously, the definition of similarity plays a major role in what is being clustered together: Shalev-Shwartz and Ben-David illustrate this with the example in Figure 4.6 [2]: given the observations represented as dots, we might be inclined to either group them in spatially separated clusters by lines (Clusters A1 and A2) or by color (Clusters B1 and B2). This emphasizes the importance of the distance or similarity metric used for clustering.

K-Means clustering is a simple and popular method to identify the clusters in data. There exist multiple variants; here, we will focus on an objective function that minimizes the distances of the data points to the mean of each cluster. Note that, similar to many other clustering methods, we require the target number of clusters K for initialization. The iterative clustering has the following algorithmic steps:

1. *Initialization*: randomly select K observations from the dataset as initialization.
2. *Update*: compute the mean for each cluster.
3. *Assignment*: assign each observation to the cluster with the mean closest to the observation.
4. *Iteration*: go to step 2 until the clusters stabilize.

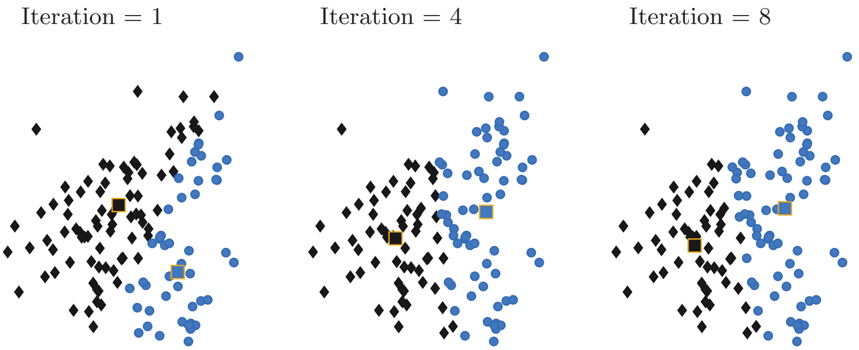
```
% assign observations to clusters
clusterIdx = assignClusterLabels_I(V, state);

for i = 1:numMaxIter
    prevState = state;

    % update means
    state = computeClusterMeans_I(V, clusterIdx, K);

    % reinitialize empty clusters
    state = reinitState_I(state, clusterIdx, K, range_V);

    % assign observations to clusters
    clusterIdx = assignClusterLabels_I(V, state);

    % if we have converged, break
    if (max(sum(abs(state.m-prevState.m))) == 0)
        break;
    end
end
```

Repo 2: ToolSimpleKmeans.m
Repo 3: ToolSimpleKmeans.py

Listing 4.3 Implementation of K-Means clustering.

Listing 4.3 shows an implementation of this algorithm.

Figure 4.7 shows the results of this implementation applied to the observations in the feature space shown in Figure 4.2 for three different stages. The algorithm converges after eight iterations. Note that in this case, the clustering based on these two features with this the distance measure is unsuccessful if the goal is to match the two ground truth classes music and speech (compare with the ground truth distribution from Figure 4.2).


Iteration $= 1$
Iteration $= 4$
Iteration $= 8$

Repo 1: plotKmeans.m

Figure 4.7 Three iterations of K-means clustering. The colors indicate the cluster assignments and the square indicates the current cluster mean.

4.4 Distance and Similarity

Distance Measures are an integral part of many algorithms introduced in this book. Whether our intention is to classify a new observation with the K-NN classifier (see Section 4.1), to find clusters in the feature space (see Section 4.3), to identify musically "similar" pieces (see Section 12.1), or to compute the most likely key in a piece of music (see Section 7.5): some form of distance between observations may need to be computed.

A large variety of distance measures exists, and different distance measures are suited to different tasks, data representations, and spaces. Deza and Deza provide an excellent overview of the different distance measures [11]. At this point, only the most widely used measures are briefly and informally introduced. Typical distance measures between the vectors $\boldsymbol{v}_\mathrm{A}$ and $\boldsymbol{v}_\mathrm{B}$ of dimensionality $\mathcal{J}$ are the following:

- *Euclidean distance*:

  $$d_\mathrm{EU}(\boldsymbol{v}_\mathrm{A}, \boldsymbol{v}_\mathrm{B}) = \|\boldsymbol{v}_\mathrm{A} - \boldsymbol{v}_\mathrm{B}\|_2 = \sqrt{\sum_{j=0}^{\mathcal{J}-1} \big(v_\mathrm{A}(j) - v_\mathrm{B}(j)\big)^2}. \tag{4.5}$$

  It is also referred to as L2 distance.
- *Manhattan distance*:

  $$d_\mathrm{M}(\boldsymbol{v}_\mathrm{A}, \boldsymbol{v}_\mathrm{B}) = \|\boldsymbol{v}_\mathrm{A} - \boldsymbol{v}_\mathrm{B}\|_1 = \sum_{j=0}^{\mathcal{J}-1} \left|v_\mathrm{A}(j) - v_\mathrm{B}(j)\right|. \tag{4.6}$$

  This distance is also known as L1 distance or rectilinear distance.
- *Cosine similarity*:

  $$s_\mathrm{C}(\boldsymbol{v}_\mathrm{A}, \boldsymbol{v}_\mathrm{B}) = \frac{\sum\limits_{j=0}^{\mathcal{J}-1} v_\mathrm{A}(j) \cdot v_\mathrm{B}(j)}{\sqrt{\sum\limits_{j=0}^{\mathcal{J}-1} v_\mathrm{A}(j)^2} \cdot \sqrt{\sum\limits_{j=0}^{\mathcal{J}-1} v_\mathrm{B}(j)^2}}. \tag{4.7}$$

  The cosine similarity does not easily fit in this list of distances because (i) its range is $[-1, 1]$ in the case of potentially negative inputs, otherwise $[0, 1]$, (ii) it is not a distance but a similarity measure, however, in the case of nonnegative inputs the *cosine distance* can be computed as follows:

  $$d_\mathrm{C}(\boldsymbol{v}_\mathrm{A}, \boldsymbol{v}_\mathrm{B}) = 1 - s_\mathrm{C}(\boldsymbol{v}_\mathrm{A}, \boldsymbol{v}_\mathrm{B}), \text{ and} \tag{4.8}$$

  (iii) it does not take into account the length of the input vectors, only the angle between them. In case the inputs $\boldsymbol{v}_\mathrm{A}, \boldsymbol{v}_\mathrm{B}$ have zero mean, the Cosine Similarity equals the *Pearson Correlation Coefficient* (see Appendix A.3).

- *Kullback–Leibler Divergence*:

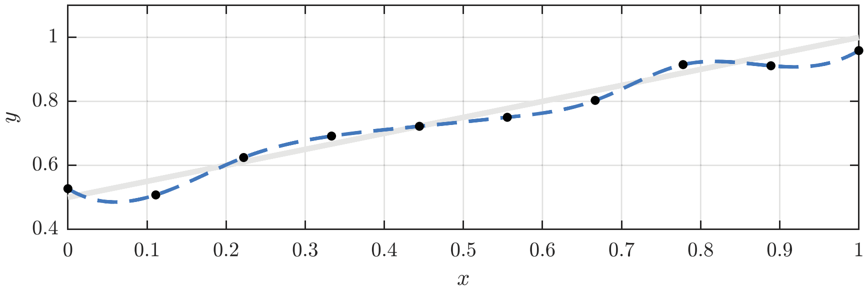
$$d_{\mathrm{KL}}(\boldsymbol{v}_{\mathrm{A}}, \boldsymbol{v}_{\mathrm{B}}) = \sum_{j=0}^{\mathcal{J}-1} v_{\mathrm{A}}(j) \cdot \log\left(\frac{v_{\mathrm{A}}(j)}{v_{\mathrm{B}}(j)}\right). \tag{4.9}$$

The Kullback–Leibler Divergence is not a great fit in this list either, as it is not symmetric, i.e. $d_{\mathrm{KL}}(\boldsymbol{v}_{\mathrm{A}}, \boldsymbol{v}_{\mathrm{B}}) \neq d_{\mathrm{KL}}(\boldsymbol{v}_{\mathrm{B}}, \boldsymbol{v}_{\mathrm{A}})$ and thus not really a distance measure. It is designed to measure how different one probability distribution is to another.

4.5 Underfitting and Overfitting

A machine-learning model can struggle to converge to a solution if the model is too simple for a complex task. A linear regression, for example, will fail to model nonlinear data as it is not able to capture the underlying structure of the data. It will thus show poor performance. This is called *underfitting* and can usually be solved by increasing the machine-learning model complexity.

Generally of more concern, however, is *overfitting*. In this case, the model works very well on the data it has been trained on but poorly on unseen data [3]. Figure 4.8 shows a case of a regression supposed to model a simple linear relationship. Because of noise in the training data, the observations are slightly off; the model “interprets” this noise as important input and adapts the trained model to match all observations exactly, which in turn leads to problems with generalization. In other words, while this model fits the training data very well, it might not perform well on unseen data. Overfitting can be most easily detected by comparing the accuracy on the training data with the accuracy on validation data over training epochs. A decreasing validation accuracy combined with an increasing training accuracy indicates overfitting. Overfitting can be typically addressed by (i) reducing the model complexity and the input dimensionality,



Repo 1: plotOverfitting.m

Figure 4.8 Example of visualization of an overfitted model exactly matching the noisy observations (dashed line) vs. a correct model (straight line) given a set of noisy data points (black).

(ii) reducing the training iterations, (iii) increasing the amount of training data, and (iv) using training strategies designed to reduce the risk of overfitting such as dropout etc.

The risk of overfitting does not only exist exclusively for machine-learning systems, although the term itself is commonly associated with ML itself. Practically, the same problem can also occur with hand-tweaked expert systems: the system designer might be tweaking the system parameters to optimize results for a small set of inputs, leading to a very good performance on the development set but a potentially bad performance on unseen input data.

4.6 Exercises

4.6.1 Questions

- Describe the difference between a 1-NN and a 3-NN classifier.
- Which of the following two algorithms is considered an unsupervised machine-learning algorithm?
  - K-Nearest Neighbor
  - K-Means Clustering
- Why is overfitting a problem? List potential causes of overfitting and ways to detect it.

4.6.2 Assignments

1. Classification
   (a) Write a function `[est_class] = k−nearestneighbor(test_data, train_data, train_label, k)` that implements a k-NN classifier based on the Euclidean distance (L2 distance). `test_data` and `train_data` are feature matrices as np arrays with the dimensions numOfFeatures × numOfObservations (the number of test and training observations can be different), `train_label` contains the class labels (dimension numOfObservations) for the training data and `k` is the number of neighbors considered. The returned vector `est_class` contains numOfObservations class labels. Think about how to deal with a multiclass scenario when there might be no clear majority, and how you solve the equal distance problem in your implementation.
2. Clustering
   (a) Implement a function `[est_cluster] = kmeans(data, k, maxIter)` that implements a simple K-Means clustering based on the Euclidean distance (L-2 distance). `data` is the feature matrix as np arrays

with the dimensions numOfFeatures × numOfObservations, `k` is the number of clusters, and `maxIter` is the maximum number of iterations before termination. The returned vector `est_cluster` contains numOfObservations cluster indices. For the initialization, pick `k` random points from your `data`.

3. Linear Regression
   (a) Write a function `[slope, offset] = linear_regression(target, input)` that implements a one-dimensional linear regression based on the MSE. `target` and `input` are feature vectors of identical dimensions numOfObservations × 1. Return the two values `slope` m and `offset` b.

References

1 Christopher M Bishop. *Pattern Recognition and Machine Learning. Information Science and Statistics*. Springer-Verlag, Berlin, 2006. ISBN 978-0-387-31073-2.

2 Shai Shalev-Shwartz, Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*, Cambridge University Press, Cambridge, 2014. ISBN 978-1-107-29801-9. https://doi.org/10.1017/CBO9781107298019.

3 Richard O Duda, Peter E Hart, and David G Stork. *Pattern Classification*. John Wiley & Sons, Inc, New York, 2nd edition, 2000. ISBN 0-471-05669-3. 25694.

4 Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

5 Andrew Ng. *Machine Learning Yearning: Technical Strategy for AI Engineers, In the Era of Deep Learning*. to be published, 2018. URL https://www.deeplearning.ai/machine-learning-yearning/.

6 Evelyn Fix and J L Hodges. Discriminatory Analysis - Nonparametric Discrimination: Consistency Properties. Technical report, University of California Berkeley, February 1951. URL https://apps.dtic.mil/docs/citations/ADA800276.

7 Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A Training Algorithm for Optimal Margin Classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA, 1992. Association for Computing Machinery. ISBN 0-89791-497-X. doi: 10.1145/130385.130401. URL https://doi.org/10.1145/130385.130401. event-place: Pittsburgh, Pennsylvania, USA.

8 Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, October 2001. ISSN 1573-0565. doi: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324.

9 John Ross Quinlan. Induction of Decision Trees. *Machine Learning*, 1(1):81–106, March 1986. ISSN 1573-0565. doi: 10.1007/BF00116251. URL https://doi.org/10.1007/BF00116251.

10 Lawrence R Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989. ISSN 00189219. doi: 10.1109/5.18626. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=18626.

11 Elena Deza and Michel Marie Deza. *Encyclopedia of Distances*. Springer-Verlag, Heidelberg, Berlin, 2009. ISBN 978-3-642-00233-5 978-3-642-00234-2. doi: 10.1007/978-3-642-00234-2. URL http://link.springer.com/10.1007/978-3-642-00234-2.

5

Data

To build a useful system for audio analysis, data are required. In the case of rule-based expert systems, data are needed to tweak parameters and validate implementation. For data-driven machine learning (ML) systems, data become even more important as the model to map input (e.g. features) to output (e.g. class labels) is learned from the data. For most supervised ML approaches, these data have thus to be annotated with task-dependent *ground truth* labels. For example, if we want to train a music genre classifier, the data annotations have to give us each song's genre. In order to train and test a machine-learning system well, data have to meet important requirements.

First, the data have to be sufficient, i.e. the dataset has to be large enough. The more complex a task is, the more complex a system needs to be, and the more training data are required. Without a sufficient amount of data, a complex system will not be able to generalize from the underlying factors in data and thus will potentially "overfit" (see Section 4.5), meaning that the model works very well on the data it has been trained on but poorly on unseen data [1]. The amount of training data becomes a crucial issue for machine learning approaches and systems based on deep neural networks, which have shown superior performance at nearly all tasks in audio content analysis but require large amounts of data for training.

Second, the labels or annotation data should not be noisy, meaning the labels should be consistent and unambiguous. This poses a problem for many tasks in audio content analysis (ACA) as user annotations are frequently subjective, implying that no absolute ground truth exists.

Third, the data have to be representative in terms of class balance (or the label distribution in the case of regression tasks). It is common that real-world data are imbalanced, e.g. baroque music may not be streamed or sold as often as popular music, although it might represent a large corpus of recordings. Whether this imbalance should be reflected by the dataset or not is a design question that ties in with the choice of system architecture and evaluation metrics – in some cases, a uniform class distribution might suit the requirements better. Note that both, an

An Introduction to Audio Content Analysis: Music Information Retrieval Tasks & Applications,
Second Edition. Alexander Lerch.
© 2023 The Institute of Electrical and Electronics Engineers, Inc. Published 2023 by John Wiley & Sons, Inc.

unchanged "real-world" class distribution and a modified class distribution can potentially lead to a biased system and potentially unwanted results.

Fourth, the data have to be representative in terms of diversity. That means that the expected variability of the input data should be well covered to enable the system to generalize. A music genre can, for instance, not be properly represented by only one artist as the system might not be able to learn general genre characteristics. Diversity is, of course, not restricted to musical genre: other potential factors might – depending on the task – include audio and production quality and environment, the artist's background, length of the recordings, recording years, tempo, musical complexity, playing techniques, proficiency of the musicians, instrumentation, and tuning, etc. This list gives only examples and is far from being exhaustive.

Fifth, nonrelevant confounding factors have to be controlled in the data: if, for example, the training data for a speech/music classifier contains studio news recordings with speech and only live recordings with audience for music, it becomes unclear whether the trained system is actually classifying music and speech or audience vs. no audience. This point is closely related to controlling the diversity in the data, although with a slightly different focus. Careful consideration of potentially impactful factors and variables can help minimizing the risk of dataset and algorithm bias.

These points emphasize the importance of careful dataset curation, but they also indicate that creating a dataset needs considerable effort. Although there is a vast amount of music data easily accessible, not all of these data can be directly used for training a machine learning system. This is partly due to copyright and licensing issues restricting the distribution and sharing of music data in the research community. The more critical issue, however, is a lack of annotation data. A system for transcribing drum events from popular music, for instance, needs expert annotations precisely marking each drum hit in terms of instrument and timing (and possibly the playing technique). Marking these individual hits is a very time-consuming and tedious task, leading to a situation in which the increasing need for annotation data due to the increasing system complexity outpaces the annotation of new data by human annotators. Given the multitude of annotations needed for various content analysis tasks, it is likely that the gap between available annotated data and required amount of data will widen in the future. This results in a growing need of systems and approaches addressing this challenge, as is reflected by an increasing research interest in this problem from various angles. Current approaches include artificially improving the amount of data, utilizing data from related tasks, reducing the requirements on data annotations, and using semi-, self-, or unsupervised systems. As the field of machine learning evolves rapidly, it is difficult to predict if any of the methods above will become standard approaches. It is clear, however, that the lack of large-scale training data will continue to impact the progress and methods applied to audio content analysis and, therefore, warrant a closer look at current approaches.

5.1 Data Split

It is common to split one big dataset into smaller subsets, the *training data*, *test data*, and often also the *validation data*. This is accomplished by *stratified sampling* to ensure that each split is similar. While "similar" can mean different things, for a classification task this usually means that the class distribution is similar in all subsets.

The training data comprise the data that the machine learning model utilizes to learn; these data are known to the system in every stage of the process. The validation data, particularly used for deep neural networks (DNNs) with a large number of (hyper-)parameters to adjust, are used for parameter tuning; the parameters used for network training are evaluated and adjusted by computing the results for the validation data. Since this is often an empirical process, experiments are often run many times on the validation set to tweak the system parameters. Thus, the validation data is implicitly known to the system. The test data are "unseen" during training and must not have any influence on the trained model or its parametrization to ensure a reliable estimate of the real-world performance of the system.

The familiarity level of the system with the different subsets also indicates the expected outcome when evaluating: we expect high performance when an evaluation is run with training data, lower performance for an evaluation with the validation data, and even lower (normal) performance for test data. The latter result, however, should always be the publicized result.

The three subsets have to be strictly separated as, for example, it would be considered cheating if inputs that have been used for training (and are thus known to the system) are reused for testing. Otherwise, the evaluation results most likely look better than they are in a real-world scenario. This requirement should also be applied to very similar inputs. For example, even if an audio file is split into multiple nonoverlapping snippets to generate multiple observations, the data split should be applied on the file level, not the snippet level. This ensures that all snippets originating from the same file are assigned to either the training or the validation or the testing subset. Similarly, genre classifiers have reported much better results if songs from the same artist or album are in both training and testing subsets [2, 3].

Note that for each individual subset, the data requirements listed above continue to apply; all subsets have to be large enough and representative, and all subset statistics and relative properties should remain similar to the overall dataset.

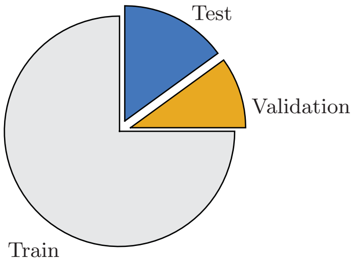


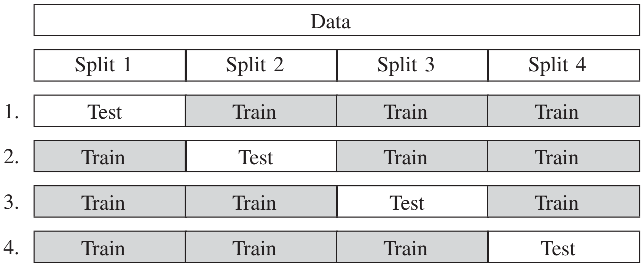
Figure 5.1 Typical percentages for splitting a dataset into train, validation, and test subsets.

Repo 1: plotDataSplit.m

A typical data split between training, validation, and testing is often 70–80%, 5–10%, and 10–20%, respectively, as shown in Figure 5.1.

5.1.1 N-Fold Cross Validation

In the case of small datasets, a fixed split into training and test data might not be feasible as the splits might not be large enough to be representative of the overall data distribution. A standard way of addressing this challenge, albeit not solving the issue of insufficient dataset size, is *N-fold cross validation*. It is a method to ensure that training data and test data are not overlapping while maximizing training and test data variability given in the data. Doing multiple train-test runs allows each observation to be in the test set once. First, the data are split into N equal-sized stratified splits. The stratification ensures that the class distribution is similar between all splits. Then, one split is designated the temporary test set and the remaining $N - 1$ splits are combined into the training data. The system is trained and tested accordingly. The process is repeated N times until each of the N segments has been used as the test set. The N evaluation results can then be averaged to receive the final evaluation result. Figure 5.2 exemplifies the data split in the case of 4-fold Cross Validation.



	Data			
	Split 1	Split 2	Split 3	Split 4
1.	Test	Train	Train	Train
2.	Train	Test	Train	Train
3.	Train	Train	Test	Train
4.	Train	Train	Train	Test

Figure 5.2 Visualization of the data splits for training and testing in the case of 4-fold Cross Validation.

A widely -used choice for the number of folds is $N = 10$, meaning that the dataset is partitioned into 10 splits. *Leave-one-out cross validation* is an extreme case where N equals the number of data observations and each test run is only executed on one single observation. Listing 5.1 shows an example code snippet of Leave-one-out cross validation. Note that high values of N might lead to considerably increased workload as the system has to be trained and tested N times. Cross validation is an important and helpful approach allowing us to work around some of the issues with insufficiently sized datasets; however, it ultimately cannot solve these issues. Therefore, additional approaches may be needed to address limited data set size.

```
% loop over observations
for o = 1:size(FeatureMatrix, 2)
    % remove current observation from ’training set’
    v_train = [FeatureMatrix(:, 1:o-1) FeatureMatrix(:, o+1:end)]’;
    C_train = [ClassIdx(1:o-1) ClassIdx(:, o+1:end)]’;

    % compute result of Nearest Neighbor Classifier given the traindata
    res = ToolSimpleKnn(FeatureMatrix(:, o)’, v_train, C_train, 1);

    conf_mat(ClassIdx(o)+1, res+1) = conf_mat(ClassIdx(o)+1, res+1) + 1;

    % if result is correct increment number of true positives
    if (res == ClassIdx(o))
        TP = TP+1;
    end
end

% compute overall (micro) accuracy
Acc = TP / length(ClassIdx);
```

Repo 2: ToolLooCrossVal.m
Repo 3: ToolLooCrossVal.py

Listing 5.1 Implementation of Leave-One-Out Cross Validation.

5.2 Training Data Augmentation

Given an insufficient amount of annotated training data, the machine learning engineer can “cheat” by artificially increasing the amount of data. This can be done either with synthesized data or by *data augmentation*, generating new training samples through processing existing audio data with task-irrelevant transformations.

The most common way of synthesizing data is rendering Musical Instrument Digital Interface (MIDI) files into audio. A system for musical instrument recognition, for example, might be trained with audio rendered from MIDI data with instrument information. While it is possible to essentially generate any amount of audio data using this method, the resulting audio files are usually very different from "real-world" recordings. They are often perceived as unnaturally clean without realistic variations in timbre, pitch, timing, and dynamics. Furthermore, they will not contain any vocals. Such differences between training data and real-world test data may lead to an underperforming system. In practice, rendered MIDI data is most successfully used either for a pretraining step followed by training with original data or as a small extension of the original training set.

Data augmentation increases the amount of available training data by applying task-irrelevant transformations to the audio data. A secondary goal of data augmentation is increasing the robustness of the system against irrelevant variables [4] by controlling these potentially confounding variables. Figure 5.3 visualizes the basic concept of data augmentation. It is important to note that the determination of which data transformations are irrelevant depend on the task. The following forms of data augmentation are most common for ACA systems:

- *Data segmentation*: Segmenting the available data is probably the most established from of data augmentation. A longer audio sample with one label might be split into shorter, possibly overlapping, segments with identical labels [5]. This can work, for example, for music genre classification as it can be assumed that the genre does not change over the time of a song. Obviously, the new segments must have a minimum length that still allows the recognition of a genre. An counter-example of a task for which this might not work well is instrument recognition; even if it is known that a specific instrument is present in the longer segment, it cannot be guaranteed to be present in all shorter segments.
- *Audio quality degradation*: Many tasks are supposed to be largely independent on the audio quality. The music genre, for example does not change whether a song is reproduced from a high-quality or a low-quality medium. A typical way of augmenting existing data in terms of audio quality is the addition of noise to the signal. This can be a simple white or low-pass filtered noise [6] or recorded

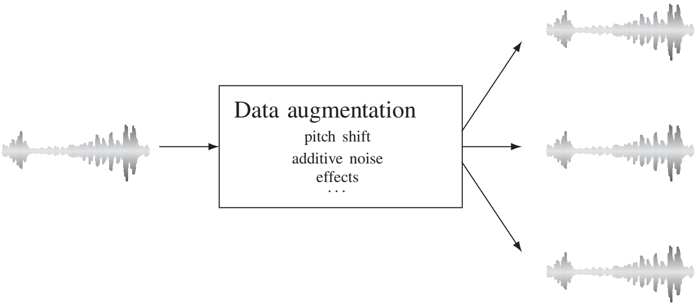


Figure 5.3 Schematic visualization of training data augmentation.

real-world noise signals such as street noise or the noise of a concert audience [7]. Other ways of impacting the audio signal quality could include simulating audio coding artifacts by encoding and decoding at different target bit rates or through quantization, clipping, or even aliasing [8].

- *Application of audio effects*: A variety of audio effects can be applied without impacting the task too much if used moderately. These include equalization (e.g. low-pass and high-pass filters) and adding reverberation. Modifying the input gain can be helpful as long as the input representation is not normalized; otherwise, a gain modification has to be time variant, e.g. through a dynamics compressor. Nonlinear effects such as wave-shaping and distortion might be used as well although they usually have considerable impact on the sound quality. Note that some of these effects can also be used to simulate audio degradation.
- *Changing pitch and/or tempo*: Time-stretching and pitch-shifting can be considered audio effects; however, they are listed here separately because of the complexity of their algorithms and the dramatic quality differences between different algorithms. If implemented carefully, they provide a musically relevant way of transforming the audio signal for augmentation, as subtle pitch or tempo modifications should not impact most tasks. Pitch shifting by small amounts has also been shown to increase robustness of classifiers against adversarial attacks [9]. Systems using a spectrogram input can also apply a simple low-quality variant of pitch shifting and time stretching by scaling and interpolating the spectrogram representation directly.
- *Data combination*: Some datasets for music transcription, instrument classification, or audio source separation include not only the mix of a song but also the individual instrument tracks. In this case, additional data might be generated by (i) mixing tracks from different songs [6] and/or (ii) applying different volume levels before mixing [10] and/or (iii) shifting tracks slightly in time before mixing [10]. Although these data modifications might not make any musical sense, they could provide additional training data keeping the network from overfitting and ultimately improving classification or separation performance.
- *Other forms of data augmentation*: Not all augmentation methods need to be musically relevant to be successful. In the case of an input spectrogram it is possible, for example, to randomly mask specific time-frequency regions so that even the same input might look slightly different each time [11].

5.3 Utilization of Data From Related Tasks

While there might be a lack of data for specific target tasks, data might be easily available for different but related tasks. *Transfer learning* is a concept applicable to neural networks that tries to transfer the knowledge learned from a related source task to the target task. This can be achieved by taking a network trained on the large source dataset, replacing the last few task-specific layers, and updating the whole network by retraining with the target data. Since the network had already been trained for a similar task, fewer data are needed to update it to the target task. If the target dataset is small, retraining the whole network might not be practical because of overfitting. In this case, only the weights of the replaced layers are updated, while the other network weights are kept frozen.

Two examples given in previous sections can fall in the category of transfer learning. Using synthesized data to pretrain a network, for example, clearly transfers the knowledge from the generated audio data to the real-world data. The main deviation from the process above is that task itself does not change, only the data characteristics. Feature learning (see Section 3.6) can be seen as a special case of transfer learning. One example in the music world was presented by Choi et al., who trained a network on an auto-tagging task and then successfully used the embedded representation of the trained network as a feature input to a simpler classifier for music genre classification [12].

Multitask learning can also be seen as a way of transferring knowledge between tasks; however, it is not usually a suitable method to address issues related to small dataset sizes. Multitask learning attempts to solve two or more tasks at the same time, thus exploiting commonalities between the tasks. This might improve the system's accuracy compared to being trained on a single task alone.

5.4 Reducing Accuracy Requirements for Data Annotation

Another possibility to solve the problem of insufficient data is to investigate, optimize, and simplify the process of data annotation. An example of doing so for tasks requiring time annotations is to *weakly* label them. For example, instead of marking the exact time positions where a specific musical instrument is present in a recording, it is possible to simply annotate that *somewhere* within a certain time span (e.g. 10 seconds), the instrument was active or not. This is referred to as a *weak label* (as opposed to detailed strongly labeled data). This allows the data collection to proceed more rapidly, in turn enabling the collection of larger datasets. Traditional machine learning approaches, however, may have to be modified to work successfully with weakly labeled data; more specifically, multi instance learning approaches are most common for this type of annotation. While in traditional ("single instance") supervised systems each instance is assigned a label (e.g. every second), multi-instance learning approaches provide only one label for a so-called "bag of instances" (e.g. one label for ten 1 s segments). Multi-instance learning has been successfully used, for example for musical instrument detection [13] and audio event detection [14].

Reducing the accuracy requirements on the ground truth might also allow for nonexperts as annotators, expanding the pool of potential annotators and enabling the crowd-sourcing of the annotation process. Invariably, this will also lead to more noise in the annotations, which is another facet that might have to be taken into account.

5.5 Semi-, Self-, and Unsupervised Learning

Semisupervised learning encompasses another group of approaches which can be useful in the case of insufficient labeled data. This is achieved by utilizing both labeled and unlabeled data during training. The unlabeled data support the model to learn a more complete representation of the data and thus possibly improves the classification results. Semisupervised learning can take various forms. Self-training iteratively trains the model on the labeled data, predicts the labels of the unseen data, and adds the predicted labeled data to the new training data [15]. It can be modified to, for example, add only the most confident predictions to the training data or weight the added data by confidence. A potential problem with self-training is that early mistakes might reinforce themselves. Another approach, consistency regularization, aims at penalizing output variability under input or model perturbations, thus allowing training with unlabeled data by enforcing that an observation is labeled consistently as the same class even for, e.g. different model training parameters [16].

The target of self- or unsupervised systems is to train the system without previously annotated training data by automatically estimating the labels or using some form of distance measure based on the data itself as training target. Generative models benefit from unlabeled data by providing more data points to build the model (such as a Gaussian mixture per class). This can, for example, be achieved with an auto-encoder-like architecture which learns a compact internal representation of data by trying to reconstruct it. This internal representation might then be used as input feature for other tasks (see also feature learning, Section 3.6). Another possibility is to utilize the outputs of multiple (simple) preexisting systems as training targets; these preexisting systems are called the teachers which train the new student system [17, 18]. Other approaches utilize sound synthesis iteratively to learn from unlabeled data [19].

5.6 Exercises

5.6.1 Questions

1. What are the implications of a very imbalanced data distribution? Discuss potential impacts on both training and evaluation. What are ways of reducing the impact?
2. Give examples of helpful considerations when splitting audio data into train, validation, and test sets.
3. Not every data augmentation method is suitable for each task. Given the task examples of Music Genre Classification and Multi-Pitch Detection, which augmentation methods can or cannot be used or only under special parametrizations?

5.6.2 Assignments

1. Write a function `[avg_accuracy, fold_accuracies, conf_mat] = cross_validate(data, gt_labels, k, num_folds)` that implements a N-fold cross validation where `num_folds` defines the number of folds, `data` is the feature matrix as np arrays with the dimensions numOfFeatures × numOfObservations, `gt_labels` are the ground truth class indices (numOfObservations). If available, use the kNN classifier implemented in Section 4.6.2 with parameter `k` as classifier for this implementation. Otherwise, follow these instructions:
    - Write a function `[est_class] = knearestneighbor(test_data, train_data, train_label, k)` that implements a k-NN classifier based on the Euclidean distance (L2 distance). `test_data` and `train_data` are feature matrices as np arrays with the dimensions numOfFeatures × numOfObservations (the number of test and training observations can be different), `train_label` contains the class labels (dimension numOfObservations) for the training data and `k` is the number of neighbors considered. The returned vector `est_class` contains numOfObservations class labels.

References

1 Richard O Duda, Peter E Hart, and David G Stork. *Pattern Classification*. John Wiley & Sons, Inc, New York, 2nd edition, 2000. ISBN 0-471-05669-3. 25694.

2 Arthur Flexer. A Closer Look on Artist Filters for Musical Genre Classification. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Vienna, 2007.

3 Arthur Flexer and Dominik Schnitzer. Effects of Album and Artist Filters in Audio Similarity Computed for Very Large Music Databases. *Computer Music Journal*, 34(3):20–28, 2010. URL http://www.mitpressjournals.org/doi/pdf/10.1162/COMJ_a_00004.

4 Rémi Mignot and Geoffroy Peeters. An Analysis of the Effect of Data Augmentation Methods: Experiments for a Musical Genre Classification Task. *Transactions of the International Society for Music Information Retrieval*, 2(1):97–110, December 2019. ISSN 2514-3298. doi: 10.5334/tismir.26. URL http://transactions.ismir.net/article/10.5334/tismir.26/.

5 Geoffroy Peeters. A Generic System for Audio Indexing: Application to Speech/Music Segmentation and Music Genre Recognition. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, page 8, Bordeaux, 2007.

6 Jan Schlüter and Thomas Grill. Exploring Data Augmentation for Improved Singing Voice Detection with Neural Networks. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 6, Malaga, Spain, 2015.

7 Brian McFee, Eric J Humphrey, and Juan P Bello. A Software Framework for Musical Data Augmentation. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 7, Malaga, Spain, 2015.

8 Matthias Mauch and Sebastian Ewert. The Audio Degradation Toolbox and its Application to Robustness Evaluation. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 6, Curitiba, Brazil, 2013.

9 Yi Qin and Alexander Lerch. Tuning Frequency Dependency in Music Classification. In *Proceedings of the International Conference on Acoustics Speech and Signal Processing (ICASSP)*, pages 401–405, Brighton, UK, 2019. Institute of Electrical and Electronics Engineers (IEEE). doi: 10.1109/ICASSP.2019.8683340. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2019/04/Qin-and-Lerch-2019-Tuning-Frequency-Dependency-in-Music-Classificatio.pdf.

10 Stefan Uhlich, Marcello Porcu, Franck Giron, Michael Enenkl, Thomas Kemp, Naoya Takahashi, and Yuki Mitsufuji. Improving Music Source Separation based on Deep Neural Networks through Data Augmentation and Network Blending. In *Proceedings of the International Conference on Acoustics Speech and Signal Processing (ICASSP)*, pages 261–265, March 2017. doi: 10.1109/ICASSP.2017.7952158.

11 Daniel S Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D Cubuk, and Quoc V Le. SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition. In *Proceedings of the Conference of the International Speech Communication Association (Interspeech)*, pages 2613–2617, Graz, Austria, September 2019. doi: 10.21437/Interspeech.2019-2680. URL http://arxiv.org/abs/1904.08779.

12 Keunwoo Choi, György Fazekas, Mark B Sandler, and Kyunghyun Cho. Transfer Learning for Music Classification and Regression Tasks. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 141–149, Suzhou, China, 2017. URL https://ismir2017.smcnus.org/wp-content/uploads/2017/10/12_Paper.pdf.

13 Siddharth Gururani, Mohit Sharma, and Alexander Lerch. An Attention Mechanism for Music Instrument Recognition. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Delft, 2019. International Society for Music Information Retrieval (ISMIR). URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2019/07/Gururani-et-al.-2019-An-Attention-Mechanism-for-Music-Instrument-Recogn.pdf.

14 Anurag Kumar and Bhiksha Raj. Audio Event Detection using Weakly Labeled Data. In *Proceedings of the International Conference on Multimedia (MM)*, MM '16, pages 1038–1047, New York, NY, USA, October 2016. Association for Computing Machinery. ISBN 978-1-4503-3603-1. doi: 10.1145/2964284.2964310. URL https://doi.org/10.1145/2964284.2964310.

15 Yunpeng Xu, Changshui Zhang, and Jing Yang. Semi-Supervised Classification of Musical Genre using Multiview Features. In *Proceedings of the International Computer Music Conference (ICMC)*, 2005.

16 Philip Bachman, Ouais Alsharif, and Doina Precup. Learning with Pseudo-Ensembles. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/hash/66be31e4c40d676991f2405aaecc6934-Abstract.html.

17 Chih-Wei Wu and Alexander Lerch. From Labeled to Unlabeled Data – On the Data Challenge in Automatic Drum Transcription. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Paris, 2018. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2018/06/Wu-and-Lerch-From-Labeled-to-Unlabeled-Data-On-the-Data-Chal.pdf.

18 Gabriel Meseguer-Brocal, Alice Cohen-Hadria, and Geoffroy Peeters. DALI: A Large Dataset of Synchronized Audio, Lyrics and notes, Automatically Created using Teacher-student Machine Learning Paradigm. In Emilia Gómez, Xiao Hu, Eric Humphrey, and Emmanouil Benetos, editors, *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 431–437, Paris, France, 2018. URL http://ismir2018.ircam.fr/doc/pdfs/35_Paper.pdf.

19 Keunwoo Choi and Kyunghyun Cho. Deep Unsupervised Drum Transcription. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Delft, Netherlands, 2019.

6

Evaluation

The system evaluation is often-neglected in text books focusing on the algorithmic system properties; however, it is a crucial part of building a system. After all, what good is a system without knowledge about whether it performs as expected? A proper evaluation aims at maximizing validity, reproducibility, and replicability of the results. Typical evaluation-related mistakes obfuscating the real-world performance of a system can be summarized as follows: (i) problems with the data, (ii) problems with the evaluation methodology, and (iii) problems with the metrics used and their application and interpretation.

The test data have to be large enough to (i) be representative of the task as outlined in Section 5, and (ii) provide sufficient sample size for trustworthy results. Otherwise, the results will neither be valid nor replicable on different data. Similarly, an improper data split that trains with data similar (or identical) to the test data will lead to an overestimation of the system performance. The evaluation methodology needs to specify clearly what is being evaluated, how the evaluation is executed, and how results are being assessed in order for a reproducible experimental setup. The wrong choice of metrics can be misleading; for example a simple accuracy measure for a classification problem with significantly imbalanced class distributions can produce a high value even if the classifier only ever predicts the majority class, thus not giving any insights on the real classifier performance.

Although there is much to say about evaluation procedures in general, we will restrict ourselves here to a summary of good practices that cover the most important aspects of evaluating a system:

1. Algorithm-agnostic evaluation procedure and metrics:
   The evaluation procedure has to be generic enough that it can properly evaluate different algorithms for the same task and should be designed before the system is built. This can be compared to test-driven development in software engineering, requiring a developer to write a complete test suite before starting to implement the code base. This also implies that selection of evaluation

An Introduction to Audio Content Analysis: Music Information Retrieval Tasks & Applications,
Second Edition. Alexander Lerch.
© 2023 The Institute of Electrical and Electronics Engineers, Inc. Published 2023 by John Wiley & Sons, Inc.

metrics has to be task-driven, not algorithm-driven, and should, if possible, be unrelated to the training metric. Reporting the training loss, for example can indicate that your system trains properly but does not necessarily imply how it performs on the task it is trained for. Although out of context, Goodhart's law is a fitting warning: "When a measure becomes a target, it ceases to be a good measure" [1].

2. Testing against different metrics and with different aggregations:
   There might be multiple different evaluation metrics that are relevant for the evaluation of a system. For example, in the evaluation of onset detection systems, it might not only be of interest how many onsets are predicted correctly (and incorrectly) but also how close they are to the ground truth onset times. Especially during the design and development phase, it can be helpful to look not only at the overall aggregated result (e.g. just one accuracy result) but investigate more deeply (e.g. checking outliers, verifying typically confused classes, analyzing if results point toward some unexpected input data characteristic).
3. Unambiguous formulation of the expected outcome:
   The expectations for system performance should be stated clearly, and best-case and worst-case performance boundaries should be established. The expectation of the *worst-case performance* can, on the one hand, be based on simple dataset characteristics. In a dataset with five equally distributed classes, a random number generator would lead to about 20% correct classifications; if the data distribution is imbalanced, then a so-called Zero-R-Classifier which simply returns the majority class for each input can provide a meaningful lower bound estimate. A more practical worst-case estimate, on the other hand, could be produced by a baseline system (most of the analysis systems detailed in this book can be considered to be baseline systems). Such a baseline system is built on often-used, well-established, and mostly simple-to-implement standard practices, for example, using a widely used feature set (e.g. MFCCs) and low-complexity classifiers such as a K-Nearest Neighbor (K-NN), Gaussian Mixture Model (GMM), Support Vector Machine (SVM), or Random Forest for an audio classification task.

   The theoretical *best-case performance* is often simply the maximum of the used evaluation metric (e.g. 100% accuracy); however, that is not always the case. A hypothetical example might be a system for audio classification that uses side information, namely chord labels estimated from the audio signal. In this scenario, the best-case performance of this setup will be reached if the chord detection works flawlessly. Thus, the best-case performance can be estimated by feeding the ground truth labels as so-called "oracle" into the system.
4. Implementing an automated, reproducible evaluation pipeline:
   The evaluation should be automated to allow quick verification of results and unproblematic evaluation of different algorithmic iterations. This enables other researchers to verify and compare results, possibly on different data.

Reproducibility should be made as easy as reasonably possible. Tasks enabling the reproducibility include (i) publication of the source code of algorithm, data processing, and evaluation pipeline, (ii) meticulous description of evaluation procedure, data split, and metrics, (iii) ensuring public accessibility of data.

5. Comparison to state-of-the-art systems:
   The comparison to the state-of-the-art is a necessity. Practice has shown, however, that simply referring to the results of another study as comparison has multiple potential issues. The most obvious problems are the usage of a different dataset or different metrics. In this case, all direct comparisons are meaningless. Other common potential issues are, for example, a different data split, different result aggregation, or a slightly different implementation of a metric. Therefore, the preferred approach to comparing to a state-of-the-art system is running the published system in the same experimental setup, with the identical data split and evaluation pipeline; unfortunately, this is only possible if the authors of the system have published source code (or a precompiled system) that is easy to use. The *Music Information Retrieval Evaluation eXchange (MIREX)*[1] has made laudable progress in providing a default evaluation setting for various MIR tasks.
6. Testing for statistical significance:
   Reporting measures on how statistically significant the results are can increase the confidence in the results. Especially when comparing results that are numerically close together, measures of statistical significance such as the p-value (see, e.g. [2]) can give insights into whether or not a particular conclusion can reliably be drawn.

6.1 Metrics

The use of appropriate metrics is crucial for estimating the performance of a system. Some of the presented metrics can also be used as loss functions for ML systems (e.g. the MSE for computing the best fit for linear regression and for evaluation of regression predictions); this might lead to the methodological problems discussed above. This section aims at briefly introducing the most common metrics used for the evaluation of classification, regression, and clustering approaches. Even though these metrics are established, widely used, and de facto standard, it is worth noting that no single metric is a good fit for all scenarios [3].

6.1.1 Classification

In the case of a binary classifier, there exist two classes (for the remainder of the section, they will be referred to as *Positive* and *Negative*). Possible outcomes are the correct classification of the positive or negative class, which will be referred to as *True Positive (TP)* or *True Negative (TN)*, respectively, or the incorrect classification, referred to as *False Positive (FP)* and *False Negative (FN)*, respectively. Incorrect predictions are thus labeled as *False* and correct predictions are labeled as *True*. To summarize, there are

---

[1] MIREX. https://www.music-ir.org/mirex. Last retrieved on 8 February 2022.

- TPs: Positives correctly identified as Positives,
- TNs: Negatives correctly identified as Negatives,
- FNs: Positives incorrectly identified as Negatives, and
- FPs: Negatives incorrectly identified as Positives.

These four values can be visualized in a *confusion matrix* as shown in Table 6.1. The confusion matrix provides an efficient way of parsing the output of a classifier. A perfect classifier has no false predictions; therefore, the number of true predictions on the diagonal will match the number of observations in the dataset and the other matrix entries are zero. For an imperfect classifier, the number of false classifications per class may indicate either a bias or preference of the classifier (e.g. if one column shows many more predictions than other columns) or highlight pairs of classes that are easily confused. This information becomes particularly useful in the case of multiclass problems. The confusion matrix may also be normalized to show relative values in percentages: in that case, each row is divided by the number of observations per class.

The *Accuracy* of a classifier is the ratio of correct (true) classifications and the overall number of data points:

$$\mathrm{Acc} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \tag{6.1}$$

The accuracy is thus the sum of the diagonal of the (unnormalized) confusion matrix divided by the number of data points. Note that this metric is not great in the case of imbalanced classes. In that case, the average of the accuracies per class

Table 6.1 Confusion matrix.

		Predicted		
		Positive	**Negative**	**Σ**
GT	**Positive**	TP True Positives	FN False Negatives	TP+FN # of GT Positives
	Negative	FP False Positives	TN True Negatives	FP+TN # of GT Negatives
Σ		TP+FP # of Predicted Positives	TN+FN # of Predicted Negatives	TP+TN # of True Predictions

Source: GT refers to the ground truth annotations.

can be computed so that each class has the same impact on the measure regardless of the number of data points per class:

$$\mathrm{Acc}_{\mathrm{Macro}} = \frac{\frac{\mathrm{TP}}{\mathrm{TP+FN}} + \frac{\mathrm{TN}}{\mathrm{TN+FP}}}{2} = \frac{\mathrm{TPR} + \mathrm{TNR}}{2} \tag{6.2}$$

The is referred to as the *macro accuracy* or *balanced accuracy* (similarly, the imbalanced accuracy can also be referred to as *micro accuracy*). The macro accuracy is the mean of the diagonal of the (normalized) confusion matrix. The ratios in the numerator are called True Positive Rate (TPR) and True Negative Rate (TNR) (similarly, we can also compute the False Positive Rate (FPR) and the False Negative Rate (FNR)). Both the accuracy and the macro accuracy range from 0 to 1 with 1 being the perfect score; they are identical for balanced class distributions.

Another common metric for binary classifiers is the so-called F-Measure. The F-Measure is the harmonic mean of the two metrics Precision and Recall. The *Precision P* measures what proportion of the predicted positives are correctly predicted:

$$P = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}. \tag{6.3}$$

The *Recall R* measures what proportion of the ground truth positives are correctly predicted:

$$R = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}. \tag{6.4}$$

The F-measure is then

$$F = 2 \cdot \frac{P \cdot R}{P + R}. \tag{6.5}$$

Similar to the accuracy, the F-Measure ranges from 0 to 1 with 1 being the perfect score. A generalization of the F-Measure is the F-beta Score, which applies a weight to the recall to potentially compensate for imbalanced data distributions or tweak for preferred outcomes (e.g. allowing a higher weight for precision).

Note that depending on the field, the same measures can have different names. We can already see from Eqs. (6.2) and (6.4) that the Recall is the TPR, but it may also be referred to as *Sensitivity*. The TNR is sometimes referred to as *Specificity* or *Selectivity*. A FP is a “Type I error” and a FN a “Type II error.” The F-measure is also named F-score or F1 score. This inconsistency in terminology sometimes complicates discussions on these metrics.

Computation of the accuracy and the F-measure requires the output of the classifier to be a binary value. Some classifiers, however, output a value between 0 and 1 representing some measure of class likelihood. This output can be thresholded to yield the expected binary output; however, the choice of the threshold significantly impacts the measured classifier performance. The Area Under Curve (AUC) is a threshold-independent metric since it evaluates the classifier performance for many thresholds. In order to do so, it measures the area under the Receiver Operating Characteristic Curve (ROC) [4], which plots the TPR over the FPR at different classification thresholds. A lower threshold leads to more positives and a higher threshold to fewer positives; the extreme boundary cases are either a threshold of 0, leading to all positives, or a threshold of 1, leading to no positives. The ROC of a random classifier is therefore a line with a slope of 1 with a resulting AUC of 0.5. The best classifier will yield a TPR of 1 regardless of the threshold or number of FPs; therefore, the AUC will equal 1. Instead of computing the AUC from the ROC (ROC-AUC), the AUC can also be computed from the precision-recall curve (PR-AUC). Figure 6.1 sketches a few prototypical ROCs and lists the resulting AUC values.

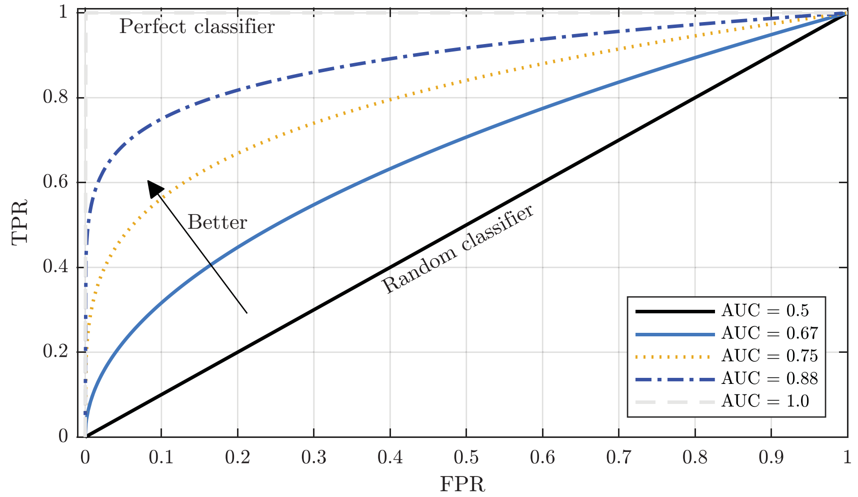
6.1.2 Regression

Since a regression algorithm predicts a value and not a category like a classifier, different metrics are necessary. More specifically, a prediction cannot be evaluated with a binary match/no-match evaluation, but by measuring some form of deviation from the ground truth. Two of the most intuitive metrics for a deviation between ground truth (target) value y and predicted value $\hat{y}$ are the Mean Absolute Error (MAE)

$$\mathrm{MAE} = \frac{1}{\mathcal{N}} \sum_{\forall n} |y(n) - \hat{y}(n)|, \quad \text{and} \tag{6.6}$$

the Mean Squared Error (MSE)

$$\mathrm{MSE} = \frac{1}{\mathcal{N}} \sum_{\forall n} \big(y(n) - \hat{y}(n)\big)^2. \tag{6.7}$$



Repo 1: plotROC.m

Figure 6.1 Multiple hypothetical ROCs and their corresponding AUCs.

The range of these metrics depends on the range of the data; a result closer to zero means better prediction. The advantage of the MAE over the MSE is that it gives the error on the scale of the data. The MSE tends to be the more widely used metric of the two.

The *Coefficient of Determination* R^2 measures the error compared to the error of a baseline prediction simply representing the mean of the ground truth data as a constant output μ_y. The R^2 is computed as follows:

$$R^2 = 1 - \frac{\mathrm{MSE}\,(y - \hat{y})}{\mathrm{MSE}\,(y - \mu_y)}. \tag{6.8}$$

While a perfect prediction leads to an R^2 of 1, it is not bounded by a minimum: very bad predictors can lead to negative R^2 results.

6.1.3 Clustering

Clustering is a special case to evaluate, as many clustering algorithms are unsupervised and thus can be applied to unlabeled data. If ground truth class labels are available, classification metrics can be applied to the results without any distinction between clustering and classification from an evaluation point of view after the cluster indices are manually mapped to the class labels. The evaluation becomes trickier, however, if no such class labels are available. A good clustering result should generally have two properties, (i) good cluster separation, i.e. maximal distance between different clusters, and (ii) good cluster cohesion, i.e. data points belonging to the same cluster are close together. The *Silhouette coefficient* is a metric combining these two properties. It is based on the so-called *silhouettes* [5]. Its calculation consists of the following steps:

1. For each data point n, compute the mean distance to all other data points of the same cluster $d_{\mathrm{intra}}(n)$.
2. For each data point n, compute the mean distance to the nearest cluster by finding the minimum of the mean distance to all data points per cluster $d_{\mathrm{neighbor}}(n)$.1
3. For each data point j, compute the *silhouette* S with

$$S(n) = \frac{d_{\mathrm{neighbor}}(n) - d_{\mathrm{intra}}(n)}{\max\,(d_{\mathrm{neighbor}}(n), d_{\mathrm{intra}}(n))}. \tag{6.9}$$

   Each resulting silhouette has a range of $[-1, 1]$.
4. The *Silhouette coefficient* is then the maximum of the mean silhouettes per cluster [6].

A different metric is named after its authors Davies and Bouldin [5]. In its simplest form, it is based on the following steps:

1. Compute the standard deviation $\sigma(k)$ for each cluster k.
2. Compute the distances $d_\mu(k_1, k_2)$ between all cluster centroids.

3. For each pair of clusters, compute
$$r(k_1, k_2) = \frac{\sigma(k_1) + \sigma(k_2)}{d_\mu(k_1, k_2)} \tag{6.10}$$
and for each cluster, take the maximum pairwise distance $r_{\max}(k_1)$.
4. Finally, the *Davies–Bouldin-Index* is average over all maximum distances.

6.2 Exercises

6.2.1 Questions

1. List typical classification metrics.
2. How can a class imbalance in the test dataset be taken into consideration when choosing evaluation metrics?
3. When talking about *True Positives* and *False Negatives*, explain what is being referred to as True/False and Positive/Negative, respectively.
4. Describe in your own words the metrics Precision and Recall and their meaning.

References

1 Marilyn Strathern. 'Improving Ratings': Audit in the British University System. *European Review*, 5(3):305–321, July 1997. ISSN 1474-0575, 1062-7987. doi: 10.1002/(SICI)1234-981X(199707)5:3<305::AID-EURO184>3.0.CO;2-4. URL https://www.cambridge.org/core/journals/european-review/article/improving-ratings-audit-in-the-british-university-system/FC2EE640C0C44E3DB87C29FB666E9AAB. Publisher: Cambridge University Press.

2 Joseph Berkson. Tests of Significance Considered as Evidence. *Journal of the American Statistical Association*, 37(219):325–335, 1942.

3 Nathalie Japkowicz. Why Question Machine Learning Evaluation Methods (An Illustrative Review of the Shortcomings of Current Methods). In *AAAI Workshop on Evaluation Methods for Machine Learning*, 2006. URL https://www.aaai.org/Library/Workshops/2006/ws06-06-003.php.

4 Tom Fawcett. An Introduction to ROC Analysis. *Pattern Recognition Letters*, 27(8):861–874, June 2006. ISSN 01678655. doi: 10.1016/j.patrec.2005.10.010. URL http://linkinghub.elsevier.com/retrieve/pii/S016786550500303X.

5 David L Davies and Donald W Bouldin. A Cluster Separation Measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):224–227, April 1979. ISSN 1939-3539. doi: 10.1109/TPAMI.1979.4766909. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

6 Leonard Kaufman and Peter J Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley series in probability and mathematical statistics. Wiley, Hoboken, NJ, 2005. ISBN 978-0-471-73578-6.

Part II

Music Transcription

7

Tonal Analysis

Tonal aspects play an important role in understanding and analyzing music, as can be seen from the vast number of pitch-related publications in music theory. Pitches are the basic building blocks of key, melody, and harmony in music.

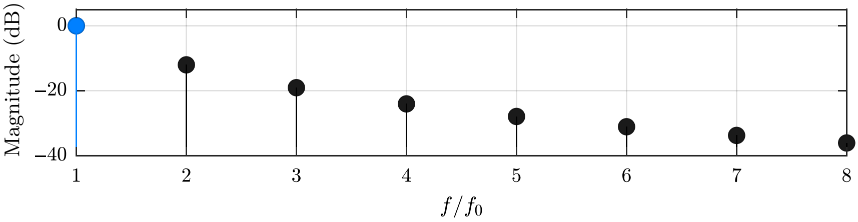
7.1 Human Perception of Pitch

The human perception of *pitch* is directly related to the frequency of a sinusoidal in the way that higher frequencies will lead to the perception of a higher pitch. If the signal is a combination of sinusoidal components with the frequencies $f_0, 2f_0, 3f_0, \dots$ (which is a reasonable approximation for pitched, quasiperiodic sounds produced by many musical instruments), then the *fundamental frequency* f_0, the frequency of the longest period, generally dominates the perceived pitch [1]. As a matter of fact, humans will usually even perceive the same pitch for this combination of harmonics if the fundamental frequency f_0 has low power or is missing [2, 3]. Figure 7.1 shows a schematic model of how the magnitudes harmonics in the spectrum of a periodic signal might be distributed. The pitch we perceive is usually related to the first harmonic at f_0, the fundamental frequency.

7.1.1 Pitch Scales

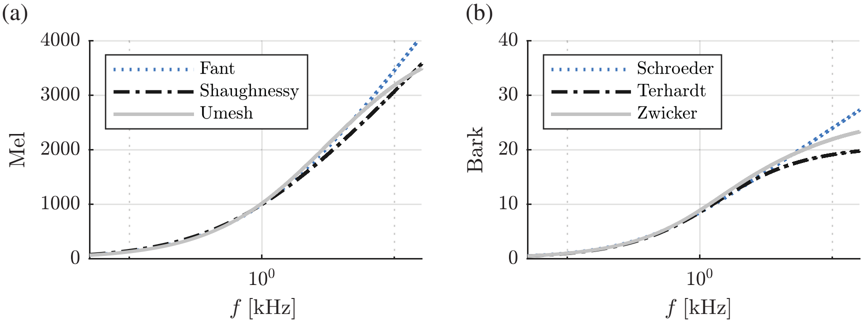
The relation between the fundamental frequency and the perceived pitch is nonlinear; at higher frequencies, two pitches with the same perceived pitch distance will have a larger frequency distance than at lower frequencies. Simply put, the nonlinearity is tied to the frequency resolution of the human cochlea. There are different approaches to measure this cochlear frequency map and thus there exist different proposals modeling it. The most common models are the *mel scale* and the *critical band rate*, also called *Bark scale* (Figure 7.2). There have been a number of proposals for analytical functions to approximate

An Introduction to Audio Content Analysis: Music Information Retrieval Tasks & Applications,
Second Edition. Alexander Lerch.
© 2023 The Institute of Electrical and Electronics Engineers, Inc. Published 2023 by John Wiley & Sons, Inc.



Repo 1: plotHarmonics.m

Figure 7.1 Prototype visualization of harmonics as integer multiples of the fundamental frequency.



Repo 1: plotMelBark.m

Figure 7.2 Different models for the nonlinear mapping of frequency to Mel (a) and Bark (b).

the measurement data resulting from listening tests for these scales. In most practical audio analysis applications, the use of one or another approximation appears to be circumstantial [4, 5]. Nevertheless, several of these models will be presented below to illustrate the variety of options.

The term *Mel* was introduced in 1937 by Stevens et al. as the name of a subjective pitch unit [6]. The *Mel scale* is a measure of tone height. The empirical data used to build the numerous analytical models of the Mel scale stem from only a limited number of psychological experiments: the most important test results have been presented by Stevens et al. [6], Stevens and Volkmann [7], and Siegel [8]. Three models will be presented here; the two older models by Fant [9]

$$\mathfrak{m}_\mathrm{F}(f) = 1000 \cdot \log_2\left(1 + \frac{f}{1000\ \mathrm{Hz}}\right) \tag{7.1}$$

and O'Shaughnessy [10][1]

$$\mathfrak{m}_\mathrm{S}(f) = 2595 \cdot \log_{10}\left(1 + \frac{f}{700\ \mathrm{Hz}}\right) \tag{7.2}$$

---

[1] Appears earlier in [11].

are most commonly used.[2] There exist multiple other models, for example the model proposed by Umesh et al. [12] is less widely known and is referenced here mainly to demonstrate the variety of models:

$$\mathfrak{m}_{\mathrm{U}}(f) = \frac{f}{2.4 \cdot 10^{-4} f + 0.741}. \tag{7.3}$$

Listing 7.1 shows the implementation of the models listed below.

```
function [mel] = acaFant_I(f)
    mel = 1000 * log2(1 + f/1000);
end

% Shaughnessy
function [mel] = acaShaughnessy_I(f)
    mel = 2595 * log10(1 + f/700);
end

% Umesh
function [mel] = acaUmesh_I(f)
    mel = f./(2.4e-4*f + 0.741);
end
```

Repo 2: ToolFreq2Mel.m
Repo 3: ToolFreq2Mel.py

Listing 7.1 Implementation of models for frequency to Mel conversion.

The *Bark scale* or *critical band rate* is constructed from the bandwidth of measured frequency groups, the critical bands. Zwicker and Fastl suggest that the Bark scale is related to the Mel scale by 1 Bark = 100 mel [13]. The most prominent models of the Bark scale have been proposed by Schroeder et al. [14]

$$\mathfrak{z}_{\mathrm{S}}(f) = 7 \cdot \operatorname{arcsinh}\left(\frac{f}{650\ \mathrm{Hz}}\right), \tag{7.4}$$

Terhardt [15]

$$\mathfrak{z}_{\mathrm{T}}(f) = 13.3 \cdot \arctan\left(0.75 \cdot \frac{f}{1000\ \mathrm{Hz}}\right), \tag{7.5}$$

and Zwicker [16]

$$\mathfrak{z}_{\mathrm{Z}}(f) = 13 \cdot \arctan\left(0.76 \cdot \frac{f}{1000\ \mathrm{Hz}}\right) + 3.5 \cdot \arctan\left(\frac{f}{7500\ \mathrm{Hz}}\right). \tag{7.6}$$

An example for a model with a lower computational complexity is Traunmüller's model [17]

$$\mathfrak{z}_{\mathrm{TM}}(f) = \frac{26.81}{1 + {}^{1960}\!/\!_{f}} - 0.53. \tag{7.7}$$

---

[2] Note that O'Shaughnessy's model is sometimes also referenced in the form

$$\mathfrak{m}_{\mathrm{S}}(f) = 1127 \cdot \log\left(1 + \frac{f}{700\ \mathrm{Hz}}\right).$$

Many more models have been proposed over the years for the nonlinear transformation of frequency to perceptual frequency groups, pitch height, and position on the human *cochlea*. Moore's model for the *Equivalent Rectangular Bandwidth (ERB)* can be seen as a model alternative to the critical band rate [18]

$$\mathfrak{e}(f) = 9.26 \log\left(1 + \frac{f}{228.7}\right). \tag{7.8}$$

Terhardt introduced a function, which he named *SPINC*, as an alternative to the Mel scale [19]:

$$\mathfrak{s}(f) = 1414 \arctan\left(\frac{f}{1414\ \mathrm{Hz}}\right), \tag{7.9}$$

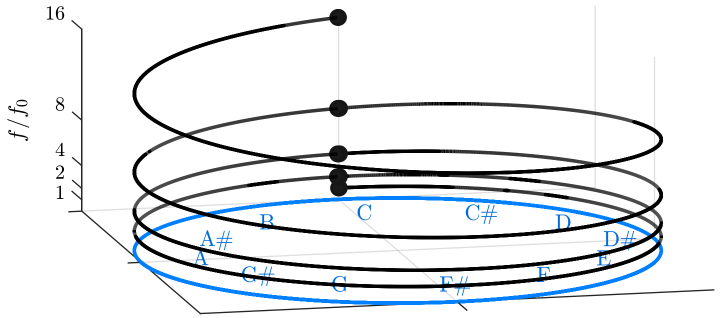
and Greenwood proposed the following equation to compute the position (normed to the range of [0, 1]) of a specific frequency on the cochlea [20]:

$$\mathfrak{x}(f) = \frac{1}{2.1}\log_{10}\left(\frac{f}{165.4} + 1\right). \tag{7.10}$$

7.1.2 Chroma Perception

There is an additional facet to human pitch perception: not only do we perceive pitch height from low to high, but we tend to group pitches with specific frequency ratios [21, 22]. More specifically, humans perceive frequencies with a frequency ratio of a power of 2 (such as $f_0, 2f_0, 4f_0, 8f_0, \ldots$) as very similar and closely related to each other. This phenomenon is usually called *chroma perception*.

Figure 7.3 visualizes this in a helix plot. On the one hand, the frequency is monotonically increasing on the z axis, modeling the tone or pitch height. On the other hand, points with the same (x, y) coordinates share a frequency ratio of a power of 2 – they share the same scale degree and are in the same pitch class (see Section 7.2.1), respectively. The circle which appears when projecting on the (x, y) plane would thus encompass all pitch classes.



Repo 1: plotPitchHelix.m

Figure 7.3 Helix visualizing the two facets of pitch perception: pitch height and chroma.

7.2 Representation of Pitch in Music

Much can be said about pitch-related properties of music, covering not only the frequency mapping of specific pitches but also interaction of pitches in chords and melodies, harmony progression, and musical key. It is not the intention of this section to give a comprehensive overview on the (music) theory involved; instead, some basics will be covered in a simplified manner since the understanding of some theoretical background can be of help in the successful design of analysis algorithms.

7.2.1 Pitch Classes and Names

The concept of musical pitch in Western music theory closely follows the pitch helix (see Figure 7.3) in that each *octave*, i.e. each range with boundaries with a frequency ratio of 2 : 1, is divided into the same chunks: the 12 *pitch classes*.

The common labels of these octave-independent pitch classes are shown in Figure 7.4. Seven of those pitch classes form the so-called *diatonic scale*. Table 7.1 shows such a diatonic scale; more specifically, the major mode (see Section 7.5) based on a root note *C* with pitch class index, pitch class name, Solfège name, and distance to the lower (scale-inherent) pitch class in semitones. As can be seen from the table, the distance between two neighboring pitches is in most cases two semitones; twice, however, it is only one semitone. Any pitch between the presented diatonic pitches can be constructed by raising the pitch (e.g. *C* → *C*♯) or lowering it (e.g. *E* → *E*♭). For now it will be assumed that, e.g. *C*♯ equals *D*♭ (a relation that is referred to as *enharmonic equivalence*), resulting in 12 pitch classes per octave as shown in Figure 7.4. The 12 pitch classes can also be visualized on a piano keyboard as shown in Figure 7.5. The white keys form the *C Major* mode shown in Table 7.1.

Figure 7.4 displays the pitch classes in one octave in musical score notation. Note that score notation includes octave information while the pitch class indices do not.

A common convention for naming musical pitches used in the following is simply the pitch class name followed by an octave index, e.g. *C*2 or *A*4. Each new octave starts with a *C* by convention.

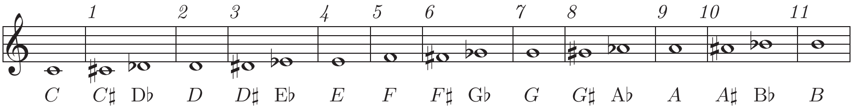


Figure 7.4 Chromatic pitches *C*4 – *B*4 in musical score notation with pitch class indices and pitch class names; enharmonically equivalent pitches are displayed twice.

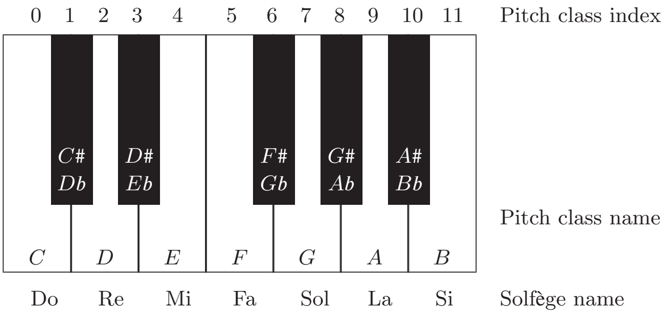

0
1
2
3
4
5
6
7
8
9
10
11
Pitch class index
C#
Db
D#
Eb
F#
Gb
G#
Ab
A#
Bb
Pitch class name
C
D
E
F
G
A
B
Do
Re
Mi
Fa
Sol
La
Si
Solfège name

Figure 7.5 One octave on a piano keyboard with annotated pitch class names.

Table 7.1 Names and distance in semitones ΔST of diatonic pitch classes.

Index	Name	Solfège Name	ΔST
0	*C*	Do	1
2	*D*	Re	2
4	*E*	Mi	2
5	*F*	Fa	1
7	*G*	Sol	2
9	*A*	La	2
11	*B*	Si	2

7.2.2 Intervals

The distance between two pitches is the musical *interval*. It is used for both the distance of simultaneously sounding pitches and pitches sounding one after another. It is a relative metric without information on absolute pitch height: humans will hear the same interval for different pairs of pitches if the ratio between their fundamental frequencies is the same. Table 7.2 names commonly used intervals and their corresponding distance in semitones.

Figure 7.6 displays the most important intervals (rising from pitch *C*4) in musical score notation.

Table 7.2 Names of musical intervals, their enharmonic equivalents, and their pitch distance in semitones.

Interval	Enharmonic Equivalent	ΔST
Unison	Diminished 2nd	0
Minor 2nd	Augmented Unison	1
(Major) 2nd	Diminished 3rd	2
Minor 3rd	Augmented 2nd	3
Major 3rd	Diminished 4th	4
(Perfect) 4th	Augmented 3rd	5
Augmented 4th	Diminished 5th/Tritone	6
(Perfect) 5th	Diminished 6th	7
Minor 6th	Augmented 5th	8
Major 6th	Diminished 7th	9
Minor 7th	Augmented 6th	10
Major 7th	Diminished Octave	11
(Perfect) Octave	Augmented 7th	12

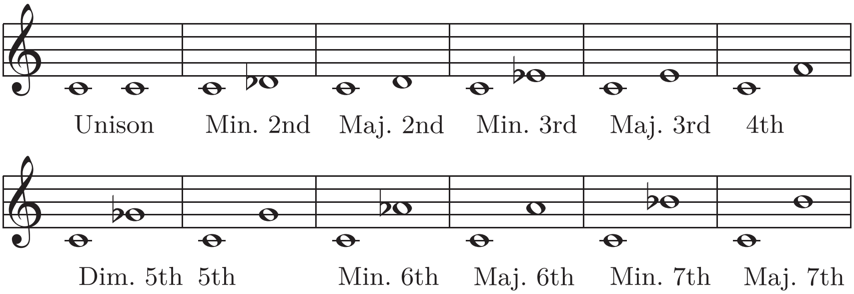


Figure 7.6 Musical intervals in musical score notation.

7.2.3 The Frequency of Musical Pitch

The most systematic model for relating musical pitch to frequency and vice versa is using the so-called equal temperament which also results in enharmonic equivalence (other temperaments will be mentioned in Section 7.2.3.1). The best example for such a transformation is the musical instrument digital interface (MIDI) scale in which each semitone has a distance of 1 to its nearest neighbor. The equations for the transformation from frequency f to MIDI pitch $\mathfrak{p}$ and vice versa are

$$\mathfrak{p}(f) = 69 + 12 \cdot \log_2\left(\frac{f}{f_{A4}}\right), \tag{7.11}$$

$$f(\mathfrak{p}) = f_{A4} \cdot 2^{\frac{\mathfrak{p}-69}{12}}. \tag{7.12}$$

The reference frequency f_{A4} is the tuning frequency (see Section 7.4); using 12 times the logarithm to the base 2 ensures that each octave is divided into 12 parts of equal "width" and the constant 69 results in the pitch $A4$ having the index 69, a convention of the MIDI standard [23] to ensure that index 0 maps to the pitch class C, and a typical piano keyboard is centered in this MIDI pitch range.[3] Thus, the MIDI pitch can be mapped easily to the pitch class index PC as introduced above by using a modulo operation:

$$\mathrm{PC}(\mathfrak{p}) = \mathrm{mod}\,(\mathfrak{p}, 12). \tag{7.13}$$

The unit Cent $\Delta C(f_1, f_2)$ is a distance measure between two pitches or frequencies f_1 and f_2. It can be computed by

$$\begin{aligned}\Delta C(f_1, f_2) &= 100 \cdot \left(\mathfrak{p}(f_1) - \mathfrak{p}(f_2)\right)\\ &= 100 \cdot \left(\left(69 + 12 \cdot \log_2\left(\frac{f_1}{f_{A4}}\right)\right) - \left(69 + 12 \cdot \log_2\left(\frac{f_2}{f_{A4}}\right)\right)\right)\\ &= 1200 \cdot \log_2\left(\frac{f_1}{f_2}\right).\end{aligned} \tag{7.14}$$

A semitone interval has thus a distance of 100 Cent and an octave has a distance of 1200 Cent.

7.2.3.1 Temperament

The temperament defines a system of frequency ratios for intervals.

In the equal-tempered case assumed above, the frequency ratio between the midfrequencies of two pitches f_1 and f_2 spaced by N semitones (with N being a negative or positive integer) is always

$$\frac{f_1}{f_2} = 2^{N/12}. \tag{7.15}$$

Therefore, the distance between two pitches is constant and independent of tonal and harmonic context; it stays also constant if the enharmonic equivalents of the two pitches are used: the frequency ratio for the interval $B, F\sharp$ is identical to the ratio of the intervals $B, G\flat$ or $C\flat, G\flat$. This, however, is only true for the equal temperament.

---

[3] Piano MIDI pitches range from 21 to 108.

The Pythagorean, meantone, and diatonic temperaments are examples of other temperaments in which the pitches are tuned depending on the key. Basically, the Pythagorean temperament is constructed from perfect fifths (frequency ratio 3 : 2), the meantone temperament is constructed with nearly perfect thirds, and the diatonic temperament intends to use as small integer ratios as possible toward the tonic for every scale degree. The pitch distances in such temperaments are, therefore, dependent on the tonic they are constructed from and an instrument tuned for one key can easily sound out of tune if played in other keys.

Table 7.3 shows the deviations in Cent of different temperaments from the equal temperament.

7.2.3.2 Intonation

In contrast to temperament, which stays fixed, the frequency of a certain pitch may also vary over time depending on the musical context. This deviation from the temperament-based frequency grid is called (expressive) intonation and is part of the musical performance.

Obviously, only musicians who are not forced to a predefined temperament can use expressive intonation. Examples are vocalists as well as players of string, brass, or woodwind instruments, while, for example, piano players have no means of adjusting the fundamental frequency of a key during a performance. It is generally assumed that musicians tend to produce pure frequency relationships such

Table 7.3 Deviations of the Pythagorean, meantone, and two diatonic temperaments from the equal-tempered scale in Cent at a reference pitch of *C*.

Pitch Class	Equally	Pythagorean	Meantone	Diatonic Major	Diatonic Minor
C	0	0	0	0	0
$C^{\#}$	0	—	—	—	—
D	0	+3.9	−6.9	+3.9	+3.9
E^b	0	—	—	—	+15.6
E	0	+7.8	−13.7	−13.7	—
F	0	−2.0	+3.4	−2.0	−2.0
$F^{\#}$	0	—	—	—	—
G	0	+2.0	−3.5	+2.0	+2.0
A^b	0	—	—	—	+13.7
A	0	+5.9	−10.2	−15.6	—
B^b	0	—	—	—	+17.6
B	0	+9.8	−17.1	−11.7	—

Source: Briner [24]/Reclam Philipp Jun.

as ${}^{f_1}\!/_{f_2} = {}^3\!/_2$ for a fifth (seven semitones) because it sounds more "natural" [25]. Furthermore, if a specific note *leads* musically to the following, the frequency will in many cases be adjusted toward the following pitch. A good example is a leading tone, where the seventh scale degree leads to the first scale degree (in *C Major*: *B* → *C*).

A special performance phenomenon related to expressive intonation is *vibrato*. Vibrato, a musical ornament, is a periodic frequency modulation of the pitch around its mean frequency. The extent and frequency of a vibrato is somewhat instrument dependent; typical frequencies are in the range of 5–9 Hz and the amplitude may be as large as two semitones [26–28].

7.3 Fundamental Frequency Detection

Fundamental frequency detection aims at detecting the fundamental frequencies in an audio recording. Identifying these frequencies enables visualization of the currently active pitches, possibly generating a score or a MIDI-like signal, tonal analysis of pitch and harmonic content, musicological analysis of intonation, vibrato, and other performance practices, and subsequent audio processing such as pitch-shifting. Fundamental frequency detection in the music domain is also frequently referred to as *pitch detection* or *pitch tracking*. Formally, the difference is that for the former, we detect the frequency in Hertz, while for the latter the musical pitch is detected. Practically, this only requires a simple conversion according to Eq. (7.11), and these terms are in fact used synonymously.

The basic assumption for all approaches to the estimation of the fundamental frequency is that the signal is periodic or quasiperiodic. The periodic state of an acoustic tone can be modeled by a Fourier series as introduced in Section 3.1.1, meaning that it is a superposition of weighted sinusoidals. The frequencies of these sinusoidals are an integer multiple of the lowest – the *fundamental* – frequency f_0. The different frequency components of a tone are called *harmonics* or *partials*, with the first harmonic being the *fundamental frequency*. Higher harmonics are also called *overtones*. While harmonics are equidistant in the frequency domain (see Figure 7.1), they are not equidistant in the pitch domain. Figure 7.7 displays the first six harmonics of the musical pitch *A*3 in traditional musical score notation.

This frequency structure can be found for most signals generated by acoustic or electronic instruments which are perceived as pitched sounds. There are, however,

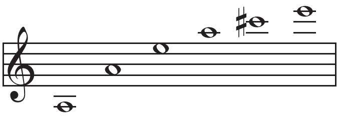


Figure 7.7 Six harmonics of the pitch *A*3 in musical score notation.

certain deviations from this rule. For example, humans will hear the fundamental frequency of a harmonic series even without this frequency actually being present in the signal [2, 3]. Although an absent fundamental frequency occurs only rarely, it can easily occur that the first harmonic has lower energy than one or more of the higher harmonics.

The piano and string instruments are examples of acoustic signals in which the harmonics do not occur precisely at integer frequency multiples of the fundamental frequency but slightly off. A model of this deviation or inharmonicity is

$$f_k = kf_0\sqrt{1 + \lambda(k^2 - 1)} \tag{7.16}$$

with k being the index of the harmonic (in this case, better called partial as it is not entirely harmonic) and λ being the inharmonicity factor with typical values in the range $[10^{-4}, 10^{-3}]$ [29]. Furthermore, there exist instruments which are perceived as being pitched yet show no clean harmonic pattern. Examples for this class of instruments are the xylophone, the vibraphone, and timpani. Nevertheless, the assumption of quasiperiodic states of a music signal has in many cases been proven to be valid and successful for fundamental frequency detection.

The common range of fundamental frequencies for musical instruments roughly starts between 20 and 50 Hz (e.g. a low pitch on the double bass) and ends between 3 and 5 kHz (e.g. a high pitch on the piccolo flute). Depending on the instrument, most of the energy is usually concentrated in three to seven lower harmonics.

7.3.1 Detection Accuracy

Algorithms for fundamental frequency detection work either in the time domain by estimating the period length of the fundamental or in the frequency domain by finding harmonic patterns. For digital signals, both are discrete value domains and have thus a maximum accuracy determined by the distance of two time domain samples and two frequency domain bins, respectively. This section provides a deeper analysis of the default error that is introduced by these discrete representations and discusses potential solutions to minimize the error.

7.3.1.1 Time Domain

As mentioned above, any estimation of the period length of the fundamental frequency is quantized to samples in the time domain, resulting in the estimated period length being a multiple of the distance between two samples

$$T_\mathrm{Q} = j \cdot T_\mathrm{S} \tag{7.17}$$

with the integer multiplier j. This quantization leads to a maximum error of $\pm T_\mathrm{S}/2$ and is thus inversely related to the sample rate. If the error in Cent is computed,

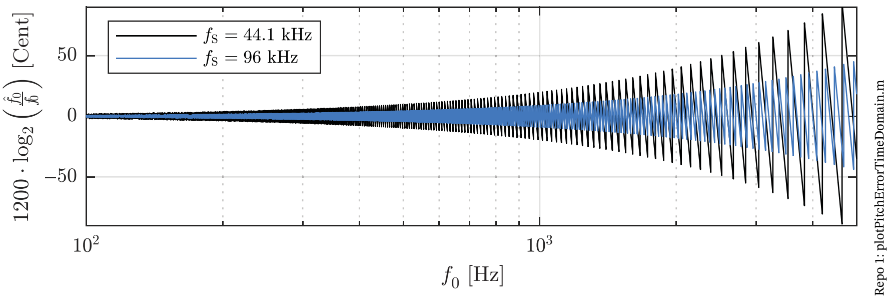


Figure 7.8 Detection error in Cent resulting from quantization of the period length to a length in samples for two different sample rates.

then it also depends on the period length of the frequency to detect. Figure 7.8 shows the minimum detection error in Cent in dependency of the fundamental frequency for two different sample rates. The absolute worst-case error is small for low frequencies and increases with the frequency as the relative impact of the quantization will be more severe for short period lengths. Obviously, higher sample rates will result in smaller errors. The sawtooth-like variation comes from the fact that some period lengths will exactly be a multiple of the time between two neighboring samples T_S and thus have no error, while others have larger errors. Note that while a detection error of a few Cent might be easily tolerable, a potential deviation of a quarter-tone or more at higher frequencies could be problematic for musical pitch detection systems.

7.3.1.2 Frequency Domain

The trade-off between time resolution and frequency resolution is one of the big issues in short time Fourier transform (STFT)-based frequency analysis. While long analysis blocks increase of the frequency resolution, they require a periodic and stationary signal during the analysis block in order to be useful (and obviously lead to a lower time resolution). As can be easily seen from Eq. (3.36), the frequency resolution is constant as long as the ratio of sample rate f_S and block length $\mathcal{K}$ stays constant:

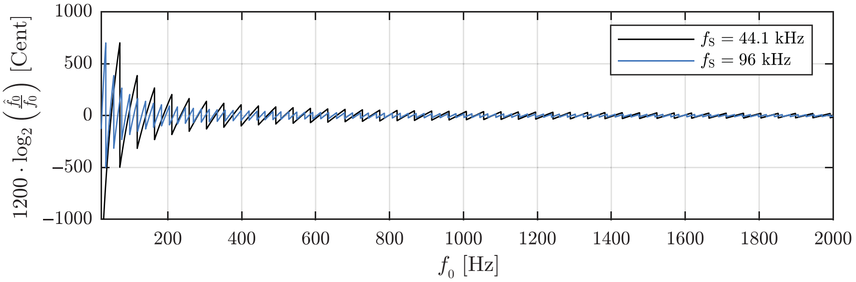
$$f_Q = k \cdot \frac{f_S}{\mathcal{K}}. \tag{7.18}$$

Table 7.4 displays the frequency resolution for different STFT sizes at a sample rate $f_S = 48$ kHz. It also shows the bin index as well as the frequency of the first bin with a bin distance larger than half a semitone.

Figure 7.9 visualizes the error in Cent for an analysis block length of 2048 samples at the sample rates 44.1 and 96 kHz. As the error is measured in Cent (logarithmic scale) it decreases with increasing frequency (linear scale). The graph

Table 7.4 Frequency resolution of the STFT for different block lengths at a sample rate of 48 kHz with bin index and frequency of the first bin k_{ST} with a distance to the following bin larger than half a semitone.

$\mathcal{K}$	Δf [Hz]	k_{ST}	$f(k_{\mathrm{ST}})$ [Hz]
256	187.50	35	6562.50
512	93.75	35	3281.25
1024	46.88	35	1640.62
2048	23.44	35	820.31
4096	11.72	35	410.16
8192	5.86	35	205.08
16384	2.93	35	102.54



Repo 1: plotPitchErrorFreqDomain.m

Figure 7.9 Detection error in Cent resulting from quantization of the fundamental frequency to the STFT bin at an analysis block length of 2048 samples for two different sample rates.

clearly highlights the problems in accuracy for low-frequency signals (note the *y*-axis scale as compared to the *y*-axis in Figure 7.8). Thus, a frequency domain algorithm that reports results at the accuracy of the frequency bin resolution can be expected to have considerable inaccuracies that may span multiple semitones.

7.3.1.3 Potential Solutions

There exist several workarounds to deal with these resolution issues. On the one hand, it is possible to combine time and frequency domain methods to try to get the best of both worlds [30], i.e. high resolution for low frequencies in the time domain and decent resolution for high frequencies in the frequency domain. On the other hand, various forms of interpolation can be used to virtually increase the resolution. As the time domain error decreases with increasing sample rate,

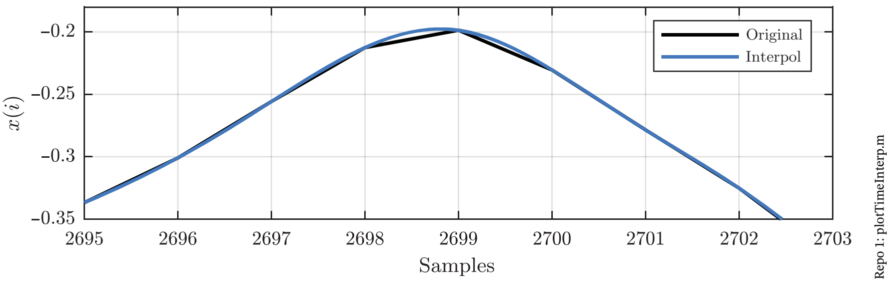
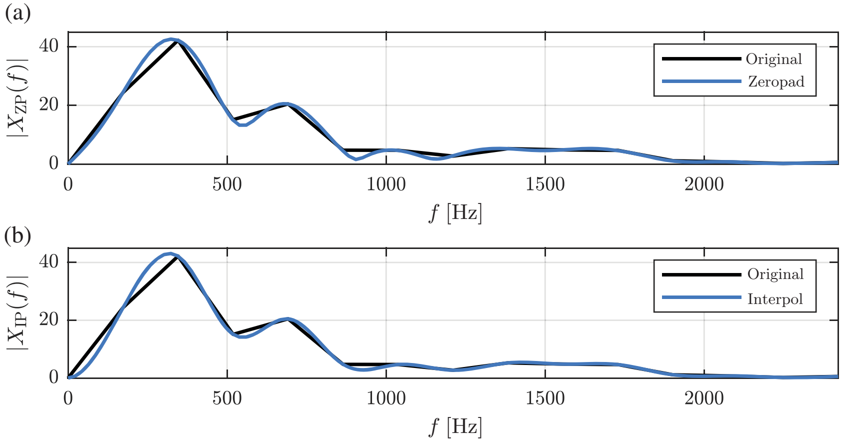


Figure 7.10 Very short excerpt of an original and a signal interpolated for peak detection.

one simple possibility is to convert the sample rate of the signal into a higher sample rate (upsampling). Done with proper postfiltering, the resulting signal may allow for a more accurate peak detection as shown in Figure 7.10. A possible disadvantage is that, if we apply upsampling to, e.g. the input of a correlation-based pitch-tracking approach (see Section 7.3.3.2), it comes at the cost of a considerable workload increase. In case this is a problem, it might also be an option to apply this upsampling step to the correlation result (or only part of it) to get a (possibly less trust-worthy) approximation of a high-resolution result.

In the frequency domain, there are two standard ways of virtually increasing the frequency resolution: (i) interpolation of the spectrum similar to the upsampling above, and (ii) applying *zero-padding* to the windowed block before computing the STFT. As the frequency resolution directly depends on the STFT block length, concatenating zeros to the windowed signal increases the frequency resolution while keeping the spectral content similar (the length difference between window length and block length will reshape the spectral leakage, however). Figure 7.11 shows both a zeropadded magnitude spectrum and a spline interpolated magnitude spectrum in comparison with the original magnitude spectrum.

Another, possibly better, way of analyzing frequency components in high resolution in the frequency domain is the computation of the *instantaneous frequency*. This approach works well if the frequency representation is mostly sparse without too many overlapping components and not too noisy. Also referred to as frequency reassignment, this method uses the phase difference between neighboring spectra to estimate the "exact" frequency of a spectral component. If applied to a spectral bin that has been identified as a candidate for the fundamental frequency, extracting the instantaneous frequency is more likely to give a more exact frequency estimate than using the bin frequency itself. The derivation and computation of the instantaneous frequency are explained in more detail in Appendix B.6.



Repo 1: plotFreqInterp.m

Figure 7.11 Magnitude spectrum and zeropadded magnitude spectrum (a) and magnitude spectrum and interpolated magnitude spectrum (b).

7.3.2 Preprocessing

Depending on the algorithmic approach, a system for fundamental frequency detection might be improved by applying appropriate preprocessing steps such as down-mixing, filtering, and sample rate conversion. It might also be helpful to remove noisy and nontonal components before the actual detection.

Since the range of fundamental frequencies is quite restricted compared to (half) the sample rates used nowadays in audio signal processing and timbre differences should be irrelevant for fundamental frequency detection systems, higher-frequency components are frequently removed by both low-pass filtering and/or down-sampling the input signal. The cut-off frequency of the low-pass filter depends on the fundamental frequency range of the input signal as well as on the usefulness of higher harmonics for the subsequent detection algorithm. The target frequency of the resampling process might also depend on the required resolution for the period length estimation (see Section 7.3.1).

In addition, a high-pass filter removing components near DC may be helpful to clean up the input signal – typical cut-off frequencies lie in the range of 30–300 Hz.

7.3.3 Monophonic Input Signals

A monophonic signal is – as opposed to a polyphonic[4] signal – single-voiced. This means there is one fundamental frequency present at a time. The problem of detecting the fundamental frequency in monophonic signals can be solved by detecting the periodicity period. The period of the first harmonic is the longest as the frequencies of the harmonics will be integer multiples of the fundamental frequency. Since many algorithms for monophonic input signals make heavy use of this property, they are of no or only limited use in the context of polyphonic input signals which are mixtures of multiple voices with possibly different and time-variant timbre.

---

[4] The term *polyphonic* will be used for signals with multiple simultaneous notes and possibly played by multiple instruments. A more restrictive definition used in musicology relates polyphony to quasiindependent voices as opposed to homophony, where the multiple voices move together in harmony.

7.3.3.1 Zero Crossing Rate

The *zero crossing rate* has already been introduced in Section 3.5.13. There are two ways to estimate the fundamental period length with zero crossings; the first is to relate the number of zero crossings in an analysis block to its length, an approach that will only produce acceptable results for very large block sizes:

$$T_0(n) = \frac{2\cdot\big(i_\mathrm{e}(n) - i_\mathrm{s}(n)\big)}{f_\mathrm{S}\cdot\sum\limits_{i=i_\mathrm{s}(n)}^{i_\mathrm{e}(n)} \left|\mathrm{sign}\,[x(i)] - \mathrm{sign}\,[x(i-1)]\right|}, \tag{7.19}$$

and the second is to measure the interval $\Delta t_\mathrm{ZC}(j)$ between neighboring zero crossings. This interval is assumed to relate directly to the fundamental period length. If $\mathcal{Z}$ zero crossings have been detected in the analysis block, the fundamental period length can be estimated with

$$T_0(n) = \frac{2}{\mathcal{Z}-1}\sum_{z=0}^{\mathcal{Z}-2}\Delta t_\mathrm{ZC}(z). \tag{7.20}$$

In case of large block lengths, the time intervals can also be sorted into a histogram. This can make the estimate more robust as spurious outliers do not influence the result. The estimated period is then the location of the histogram maximum multiplied by a factor of 2.

```
for n = 1:iNumOfBlocks
    i_tmp = diff(find(x_b(n, 1:end-1) .* x_b(n, 2:end) < 0));
    %  average distance of zero crossings indicates half period
    T_0(n) = 2 * mean(i_tmp); % or histogram max, ...
end

% convert to Hz
f_0 = f_s ./ T_0;

% avoid NaN for silence frames
f_0 (isnan(f_0)) = 0;
```

Repo 2: PitchTimeZeroCrossings.m
Repo 3: PitchTimeZeroCrossings.py

Listing 7.2 Estimation of the fundamental period length of one block of audio with zero crossings.

Listing 7.2 shows the simplicity of estimating the fundamental frequency with zero crossings. The computational efficiency does, however, come at the cost

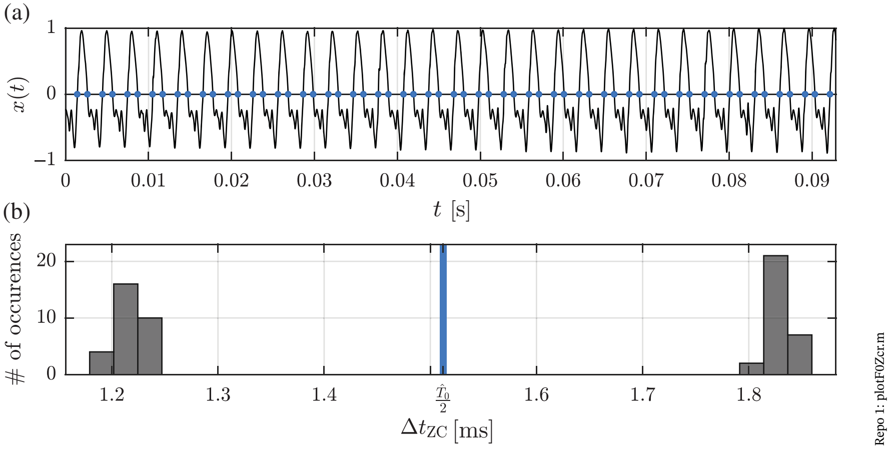


Figure 7.12 F0 estimation via the zero-crossing distance. Time series (a) and histogram of the zero-crossing distances (b) with highlighted estimated fundamental period.

of unreliable results, especially for signals with a large number of harmonics and/or noise. Figure 7.12 visualizes one of the problems; even for a block of a signal with very clearly identifiable fundamental frequency, the histogram of Inter-Zero-Crossing-Distances Δt_{ZC} does not show one peak at half the period length as expected but two peaks with very different distances. The reason for these two peaks is that the input signal is not symmetric around 0. In this particular case, averaging all distances actually results in a roughly correct fundamental period estimate as indicated by the line in the middle, however, clearly this approach does not seem to be as robust as hoped.

This problem can be somewhat addressed by applying a low-pass filter to the input signal. It is also possible to investigate the distance not only between zero crossings but also between local extrema such as the local maxima and local minima. This can increase robustness as explained by Rabiner and Schafer [31].

7.3.3.2 Autocorrelation Function

The normalized ACF (see Appendix A.3 and Section 3.5.12) is probably the most widely used approach to fundamental period length estimation. As the ACF indicates the self-similarity (i.e. the periodicity) of the signal, the lag of the maximum value is a direct estimation of the fundamental period length. Certain restrictions to maximum detection as exemplified in Section 3.5.12 can be applied to increase the algorithm's reliability. Listing 7.3 shows the implementation of an ACF-based pitch extraction with some additional constraints. Figure 7.13 shows the result of this ACF-based fundamental frequency estimation for an example audio block.

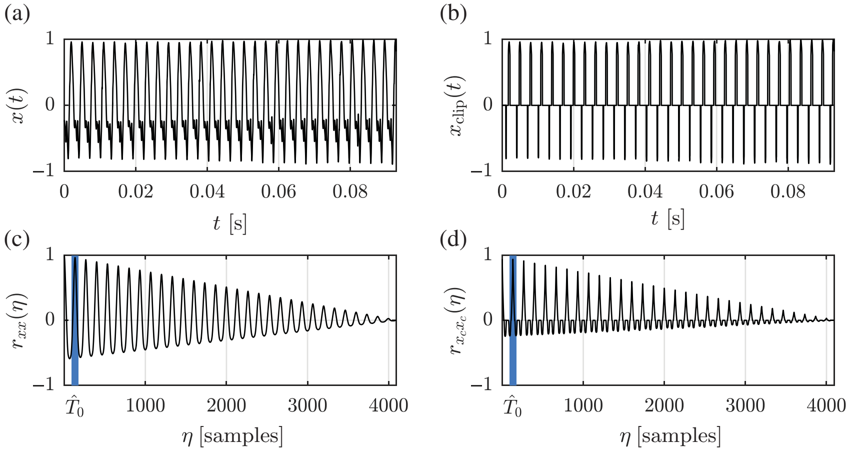
```
    for n = 1:iNumOfBlocks
        eta_min = round(f_s / fMaxFreq);

        if (sum(abs(x_b(n, :))) == 0)
            f_0(n) = 0;
            continue;
        end
        % calculate the acf maximum
        afCorr = xcorr(x_b(n, :), 'coeff');
        afCorr = afCorr((ceil((length(afCorr)/2))+1):end);

        % ignore values until threshold was crossed
        eta_tmp = find (afCorr < fMinThresh, 1);
        if (~isempty(eta_tmp))
            eta_min = max(eta_min, eta_tmp);
        end

        % only take into account values after the first minimum
        afDeltaCorr = diff(afCorr);
        eta_tmp = find(afDeltaCorr > 0, 1);
        if (~isempty(eta_tmp))
            eta_min = max(eta_min, eta_tmp);
        end

        [fDummy, T_0(n)] = max(afCorr(1+eta_min:end));

        % convert to Hz
        f_0(n) = f_s ./ (T_0(n) + eta_min);
    end
```

Repo 2: PitchTimeAcf.m
Repo 3: PitchTimeAcf.py

Listing 7.3 Source code for the extraction of the fundamental frequency from a block of audio samples with the ACF.

Low-pass filtering the input signal can improve the robustness of the output considerably. Other preprocessing steps that have been proposed are, for example:

- *Pre-whitening*:
  If the analyzed signal can be assumed to originate from a pulse-like excitation signal filtered with a transfer function, a common assumption in speech signal processing, a reasonable approach is to reverse the filtering process by estimating the smoothed spectral envelope of the current analysis block and apply the inverse filter to the signal – *pre-whitening* [32, 33]. The goal of the inverse filtering process is to convert the signal back to the initial pulse train in order to improve the results of the following correlation analysis by reducing the impact of the signal timbre. There exist numerous ways to estimate the spectral envelope, including a low-order linear predictive filter [34], an averaged power spectrum [35], or cepstrum-based iterative methods [36].



Repo 1: plotF0Acf.m

Figure 7.13 F0 estimation via the (center-clipped) Autocorrelation Function. Time series (a), center clipped time series (b), Autocorrelation Function with fundamental period T_0 (c), Autocorrelation Function of the center-clipped signal with highlighted fundamental period T_0 (d).

- *center clipping*:
  The robustness of the detection can potentially be improved by using so-called *center clipping* [31], where the nonlinear function χ as shown in Figure 7.14a is applied to the input signal $x(i)$:

  $$x_c(i) = \chi\left(x(i)\right). \tag{7.21}$$

  The idea of center clipping is to yield a result with more easily identifiable peaks. Typical thresholds c_L are chosen between 30% and 60% of the (instantaneous) maximum amplitude. An alternative three-level center clipping function χ' is shown in Figure 7.14b. This function allows a very efficient computation of the ACF because the input signal then only has the three possible amplitudes $-1, 0, +1$. This is a performance optimization, not a way to improve robustness, and is usually unnecessary on modern hardware.

7.3.3.3 Average Magnitude Difference Function

The *Average Magnitude Difference Function (AMDF)* is similar to the ACF but avoids the use of multiplications. It is computed as described by Ross et al. [37]

$$\text{AMDF}_{xx}(\eta, n) = \frac{1}{i_e(n) - i_s(n) + 1} \sum_{i=i_s(n)}^{i_e(n)-\eta} |x(i) - x(i+\eta)|. \tag{7.22}$$

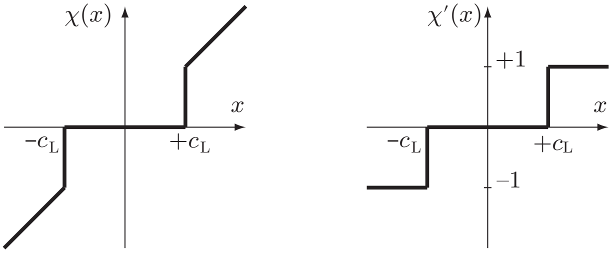


Figure 7.14 Nonlinear preprocessing for ACF-based pitch period estimation: standard center clipping (a) and three-level center clipping (b).

The estimated fundamental period length is then chosen to be the lag of the overall minimum if its value is also smaller than a certain (signal adaptive) threshold. The popular pitch-tracking algorithm YIN utilizes a variant of the AMDF [38].

The AMDF can also be combined with the ACF in an AMDF-Weighted Autocorrelation Function. There are indications that this weighting leads to more robust results [39]:

$$r'_{xx}(\eta, n) = \frac{r_{xx}(\eta, n)}{\mathrm{AMDF}_{xx}(\eta, n) + 1}. \tag{7.23}$$

Figure 7.15 visualizes the resulting AMDF (left) as well as the AMDF-weighted ACF (right) with the resulting fundamental frequency estimates. Note that in the case of this simple, untweaked implementation, the AMDF incorrectly returns a

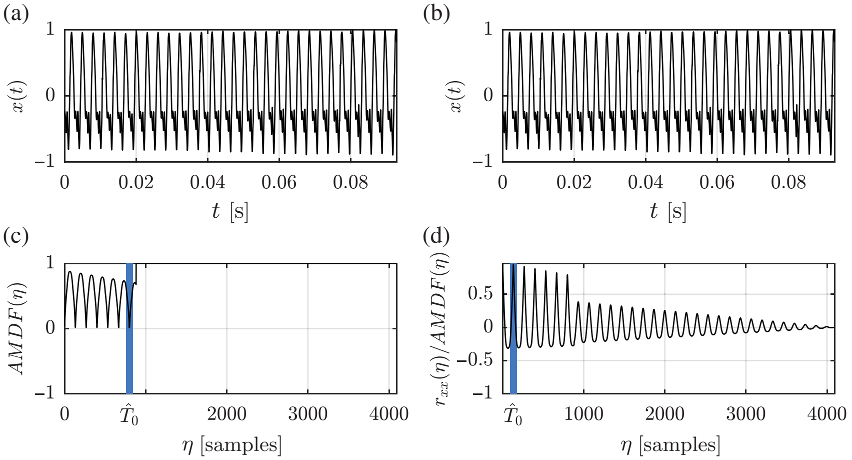


Figure 7.15 F0 estimation via the Average Magnitude Difference Function (left) and the AMDF-weighted Autocorrelation Function (right). Top: time series, Bottom: one-sided ACF with highlighted estimated fundamental period T_0.

multiple of the correct fundamental frequency. With the weighting, however, the correct fundamental frequency is estimated.

Listing 7.4 shows the implementation of computing the AMDF from signal blocks.

```
    for n = 1:iNumOfBlocks

        if (sum(abs(x_b(n, :))) == 0)
            f_0(n) = 0;
            continue;
        end

        % calculate the amdf_I minimum
        afAMDF = amdf_I(x_b(n, :), eta_max);
        [fDummy, T_0(n)] = min(afAMDF(1+eta_min:end));

        % convert to Hz
        f_0(n) = f_s ./ (T_0(n) + eta_min);
    end
```

Repo 2: PitchTimeAmdf.m
Repo 3: PitchTimeAmdf.py

Listing 7.4 Source code for the extraction of the fundamental frequency from a block of audio samples with the AMDF.

7.3.3.4 Harmonic Product Spectrum and Harmonic Sum Spectrum

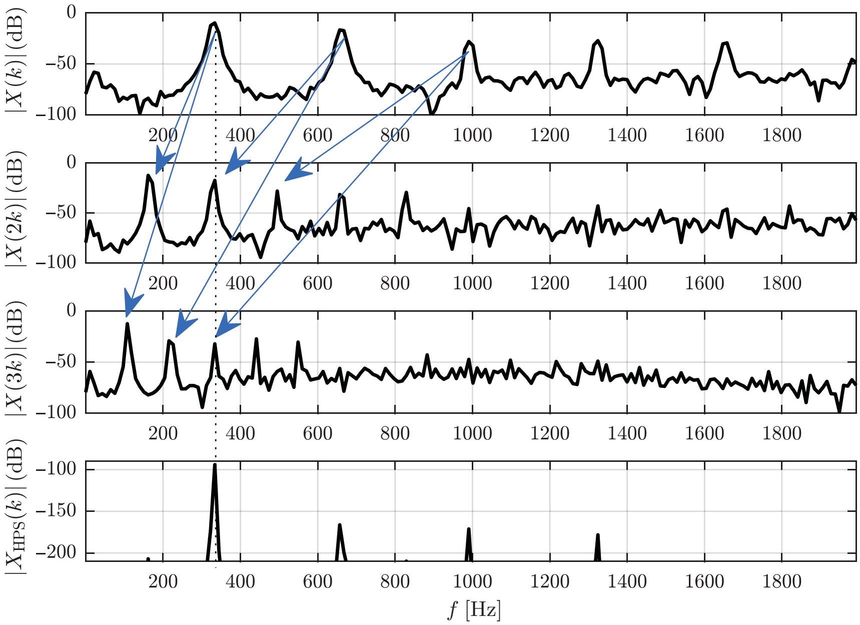
For detecting the fundamental frequency, the frequency domain has the disadvantage of insufficient frequency resolution for low frequencies (see Section 7.3.1.2) but has the advantage of being a more accessible and intuitive representation for pitch tracking: it is (for tonal signals) a sparse representation with clear peaks at the harmonics and the frequency axis allows direct access to the target value. The most trivial way of estimating the fundamental frequency from the magnitude spectrum is picking the maximum value. This is, however, a very unreliable way of estimation as the fundamental frequency is often not the most salient harmonic (see p. 139). Therefore, most frequency-domain methods for fundamental frequency estimation try to take advantage of the comb-like structure of the spectrum of a pitched signal.

The *Harmonic Product Spectrum (HPS)* is an efficient method for finding this periodic harmonic pattern [40, 41]. It is defined by

$$X_{\mathrm{HPS}}(k,n) = \prod_{j=1}^{\mathcal{O}} |X(j \cdot k, n)|^2. \tag{7.24}$$

The parameter $\mathcal{O}$ is the order of the HPS. Alternative implementations of the HPS use the magnitude spectrum.

The idea of the HPS is that the compression of the frequency axis by integer factors j causes higher harmonics at multiples of the fundamental frequency to



Repo 1: plotF0HpsMethod.m

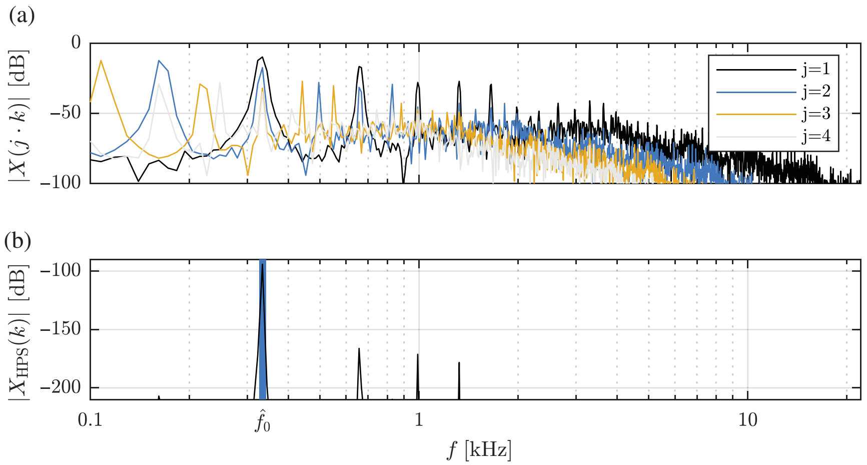
Figure 7.16 Original power spectrum (top) and compressed spectra (mid) for the computation of the HPS (bottom).

coincide at the bin of the fundamental frequency. Thus, the first $\mathcal{O}$ harmonics will be mapped to their fundamental frequency. Since the harmonics can be assumed to have significantly higher power than any other signal components, the resulting (harmonic product) spectrum $X_{\mathrm{HPS}}(k, n)$ should have a clearly identifiable peak at the fundamental frequency. Figure 7.16 visualizes the compression of the spectrum and how it leads to the higher harmonics mapped onto lower harmonics.

Figure 7.17 exemplifies this: the resulting peak in the HPS has a significantly higher distance to the second highest peak than in the original spectrum. Listing 7.5 shows the implementation of fundamental frequency extraction with HPS.

While being efficient and algorithmically “neat,” the HPS has some drawbacks. If the fundamental frequency of the input signal is not located exactly on a frequency bin but between two bins, then the maxima of higher-order harmonics will not be taken fully into account for the decimated spectra. The likelihood of missing the locations of the maxima increases with j. Two possible work-arounds can be used, but both will result in less efficient and more complicated implementations:

- increase the frequency resolution by using longer STFT block sizes or by interpolating (up-sampling) the spectrum, or



Repo 1: plotF0Hps.m

Figure 7.17 F0 estimation via the Harmonic Product Spectrum: compressed spectra with $j = 1, 2, 3, 4$ (a) and resulting HPS (b).

```
afHps = X;

% compute the HPS
for j = 2:iOrder
    afHps = afHps .* [X(1:j:end, :); zeros(size(X, 1)-size(X(1:j:end,:),
          1), size(X, 2))];
end

% find max index and convert to Hz
[fDummy, f_0] = max(afHps(k_min:end, :), [], 1);
f_0 = (f_0 + k_min - 2) / (size(X, 1)-1) * f_s / 2;
```

Repo 2: PitchSpectralHps.m
Repo 3: PitchSpectralHps.py

Listing 7.5 Source code for the extraction of the fundamental frequency from a magnitude spectrum $X(k, n)$ with HPS.

- take the maximum within a bin range for the multiplication. The bin range will have to increase by ±1 bin with every increment of j.

If one of the harmonics is zero or near zero, the detection of the fundamental frequency will fail as the HPS at the fundamental frequency bin will be scaled with (near) zero. One approach to avoid this is to compute the *Harmonic Sum Spectrum (HSS)* with a definition similar to the HPS [41]:

$$X_{\mathrm{HSS}}(k, n) = \sum_{j=1}^{\mathcal{O}} |X(j \cdot k, n)|^2. \tag{7.25}$$

While the HSS is more robust against missing harmonics, the resulting maximum is usually not as pronounced as in the HPS.

The cumulative nature of both HPS and HSS frequently leads to octave errors.

7.3.3.5 Autocorrelation Function of the Magnitude Spectrum

The usage of the ACF is, as pointed out in other sections (see Sections 3.5.12, 7.3.3.2 and Appendix A.3), a useful and intuitive approach to finding periodicities. Since the harmonics are equally spaced in the magnitude spectrum and the distance of neighboring harmonics equals the fundamental frequency, finding this periodicity is equivalent to finding the fundamental frequency. The lag of the maximum of the ACF is therefore an estimate of the fundamental frequency in spectral bins.

Figure 7.18 shows the result of the ACF of a magnitude spectrum. Listing 7.6 shows the implementation of fundamental frequency extraction with the ACF of the STFT.

The robustness of the detection can be slightly improved by computing a circular correlation of the symmetric spectrum and by manually adding a maximum at frequency bin 0 to fill a potential "gap" in the harmonic series.

7.3.3.6 Cepstral Pitch Detection

Cepstral pitch detection is based on the assumption that the analysis signal $x(i)$ is the result of a convolution of an excitation signal $e(i)$ with a transfer function $h(i)$

$$x(i) = e(i) * h(i). \tag{7.26}$$

This is a common model in speech signal processing; the excitation signal originates from the air streaming through the *glottis*. This excitation signal $e(i)$ consists of quasiperiodic pulses in the case of voiced (tonal) sounds [31]. The vocal tract and the nasal tract act as tubes that shape the frequency spectrum with the transfer function $h(i)$.

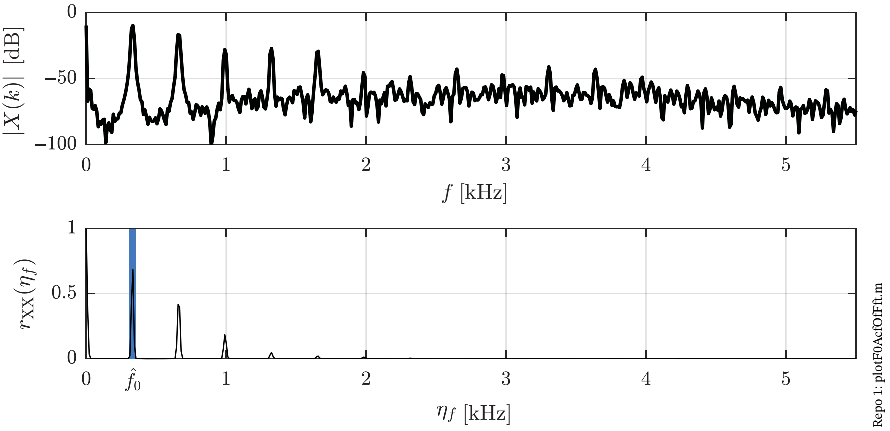


Figure 7.18 F0 estimation via the Autocorrelation Function of the magnitude spectrum: input magnitude spectrum (a) and resulting ACF (b).

```
% use spectral symmetry for robustness
X(1, :) = max(max(X));
X = [flipud(X); X];

% compute the ACF
for n = 1: size(X, 2)
    eta_min = round(f_min/f_s * (size(X, 1)-2));
    afCorr = xcorr(X(:,n),'coeff');
    afCorr = afCorr((ceil((length(afCorr)/2))+1):end);

    % find local maxima
    [fDummy, eta_peak] = findpeaks(afCorr);

    eta_min = max(eta_min, find(eta_peak > eta_min, 1));
    [fDummy, f_0(n)] = max(afCorr(eta_min:end));
end

% find max index and convert to Hz (note: X has double length)
f_0 = (f_0 + eta_min - 1) / (size(X, 1)-2) * f_s;
```

Repo 2: PitchSpectralAcf.m
Repo 3: PitchSpectralAcf.py

Listing 7.6 Source code for the extraction of the fundamental frequency from a STFT with the ACF.

Equation (7.26) can be rephrased in the frequency domain (see Appendix B.1.3) as

$$X(\mathrm{j}\omega) = E(\mathrm{j}\omega) \cdot H(\mathrm{j}\omega). \tag{7.27}$$

If a (complex) logarithm is applied to this equation, the result is

$$\begin{aligned}\log\left(X(\mathrm{j}\omega)\right) &= \log\left(E(\mathrm{j}\omega) \cdot H(\mathrm{j}\omega)\right)\\ &= \log\left(E(\mathrm{j}\omega)\right) + \log\left(H(\mathrm{j}\omega)\right).\end{aligned} \tag{7.28}$$

Applying the logarithm thus “replaces” the multiplication with an addition. The *cepstrum* $c_x(i)$ is then defined to be the inverted logarithmic spectrum

$$\begin{aligned}c_x(i) &= \mathfrak{F}^{-1}\left\{\log\left(X(\mathrm{j}\omega)\right)\right\}\\ &= \mathfrak{F}^{-1}\left\{\log\left(E(\mathrm{j}\omega)\right) + \log\left(H(\mathrm{j}\omega)\right)\right\}\\ &= \mathfrak{F}^{-1}\left\{\log\left(E(\mathrm{j}\omega)\right)\right\} + \mathfrak{F}^{-1}\left\{\log\left(H(\mathrm{j}\omega)\right)\right\}.\end{aligned} \tag{7.29}$$

This inverse-transformed spectrum consists of signals $e(i)$ and $h(i)$ being *added*, albeit after the logarithm has applied to them in the frequency domain, instead of being *convolved*. In order to emphasize this difference between the original time domain and the cepstral domain, the term *quefrency* has been suggested to be used as axis label.

The cepstrum can be approximated by only using the magnitude spectrum

$$\hat{c}_x(i_\mathrm{s}(n) \dots i_\mathrm{e}(n)) = \sum_{k=0}^{\mathcal{K}/2-1} \log\left(|X(k,n)|\right) e^{\mathrm{j}ki\Delta\Omega} \tag{7.30}$$

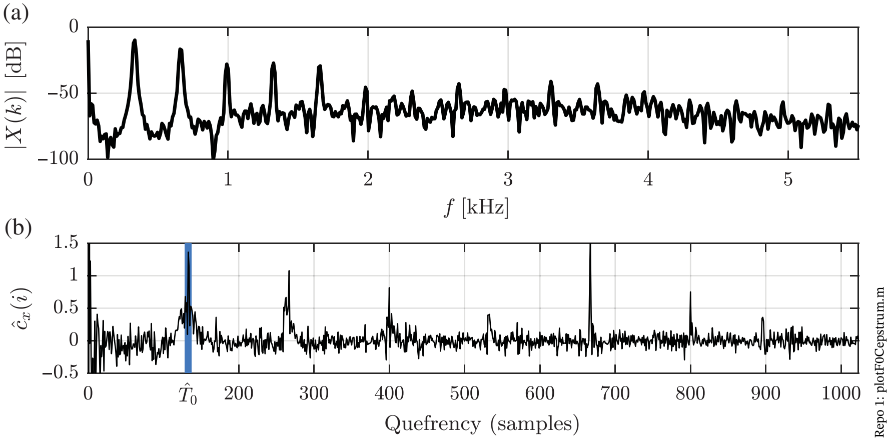


Figure 7.19 F0 estimation via the Cepstrum: input magnitude spectrum (a) and cepstrum of this magnitude spectrum (b).

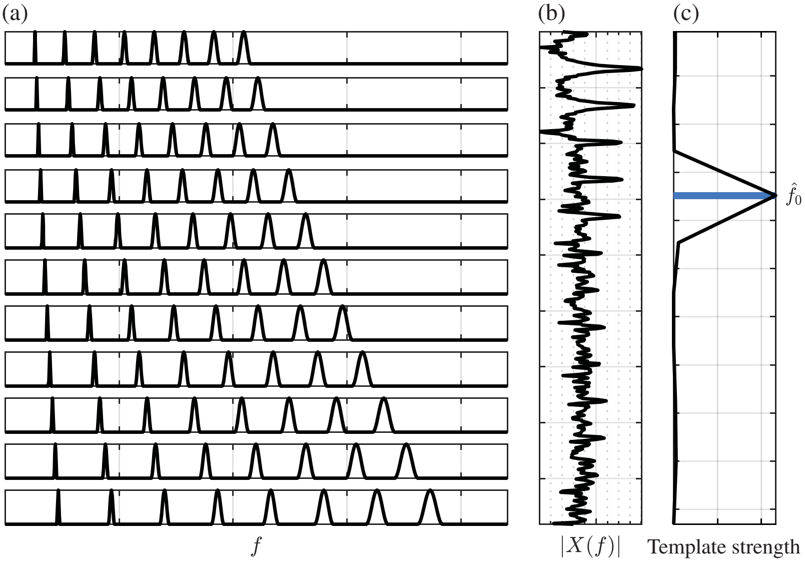
to avoid the use of the complex logarithm.

The cepstrum has two properties that are of particular interest in the context of pitch detection [40, 42]. First, a pulse-like excitation signal will also lead to a pulse in the cepstrum, and, second, the h part of the cepstrum will decay rapidly for large i. Therefore, the detection of a peak distance (or more accurately the periodicity of a pulse train) in the cepstrum should indicate the period length of the fundamental frequency. Figure 7.19 shows the cepstrum of an exemplary magnitude spectrum.

7.3.3.7 Maximum Likelihood and Template Matching

The fundamental frequency can also be detected by building a template model that closely matches the audio to be analyzed, and then using the model parameter of the fundamental frequency as a result. There have been a multitude of different proposals to do this, many of which work in the frequency domain. Figure 7.20 provides a visualization of a basic template matching approach: on the left is a matrix with a predefined set of spectral magnitude templates (the spectrum of one pitch per row), which is multiplied with the input audio magnitude spectrum in the middle. The resulting vector, plotted on the right, can be seen as the correlation value per template and the index of the maximum in this vector indicates the index of the template most closely matching the audio spectrum.

Although for the shown example block, the fundamental frequency is detected correctly, this model is likely too simple to provide reliable estimates as all the harmonics have identical weights. Another disadvantage is the quantization of the fundamental frequencies in the template that does not allow for frequencies not exactly matching the template frequencies. Therefore, approaches have



Repo 1: plotF0Template.m

Figure 7.20 F0 estimation templates: template functions with different pitches (a), example input spectrum (b), and resulting template strength (c).

been proposed to adaptively find the best set of parameters for identifying the fundamental frequency [43, 44].

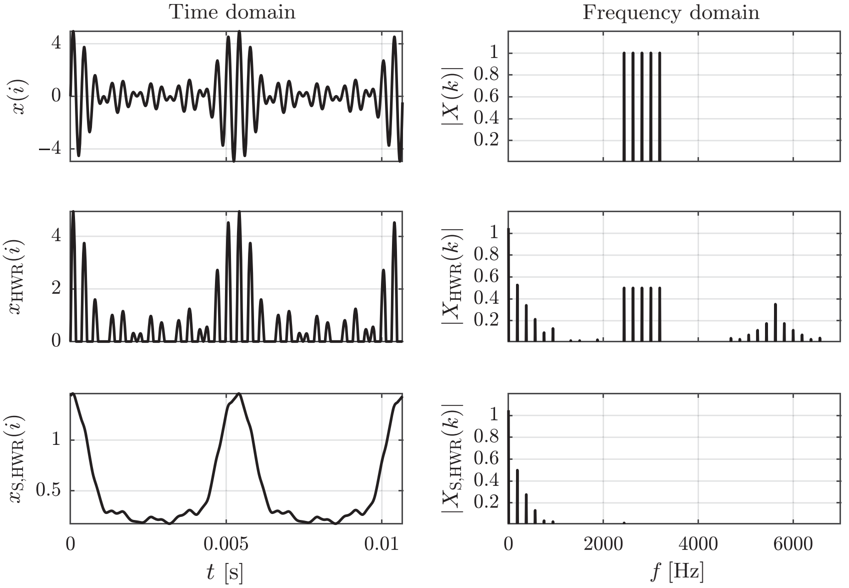
7.3.3.8 Auditory-Motivated Pitch Tracking

A class of pitch detection algorithms use models of human pitch perception to determine the pitch of a signal. Meddis and O'Mard describe the processing stages of such algorithms as [45]:

(i) band-pass filtering,
(ii) HWR [see Eq. (3.59)] and band processing,
(iii) within-band periodicity extraction, and
(iv) across-band aggregation of periodicity estimates.

The filterbank used for band-pass filtering can, for instance, be a gammatone filterbank, one of the "standard" filterbanks for auditory processing. Gammatone filters have been introduced in Section 3.3.4.

Klapuri pointed out the importance of HWR on the filter channel outputs for fundamental pitch estimation [46]. Figure 7.21 shows this effect for one band containing the 13th–17th harmonics of a periodic sound: HWR results in new frequency components being generated at the distance of frequency components in the signal and thus around the fundamental frequency. Subsequent low-pass filtering of this band output removes the (original and new) high-frequency



Repo 1: plotF0Auditory.m

Figure 7.21 Time domain (left) and frequency domain (right) visualization of a signal consisting of the 13th to 17th partial of a sound with a fundamental frequency of 187.5 Hz (top), the corresponding signal subjected to HWR (mid), and the low-pass filtered signal subjected to HWR (bottom).

components. Further band processing might include gain compression by, for instance, scaling the variance to unity [46].

A periodicity analysis of the filterbank outputs $z_c(i)$ may then be used for the detection of the fundamental period length by, for example, applying an ACF to each channel

$$r_{zz}(c, n, \eta) = \sum_{\eta=0}^{\mathcal{K}-1} z_c(i) \cdot z_c(i + \eta) \tag{7.31}$$

and summing the resulting ACFs

$$r_{\mathrm{A}}(n, \eta) = \sum_{c=0}^{\mathcal{C}-1} r_{zz}(c, n, \eta). \tag{7.32}$$

7.3.4 Polyphonic Input Signals

Most of the algorithms for fundamental frequency detection outlined above will not work well in the case of polyphonic signals with multiple simultaneous fundamental frequencies. The expected number of simultaneous pitches in polyphonic signals depends on genre, epoch, and musical context. In most cases, the number of independent voices will be between one and eight.

The performance of multi-pitch detection systems has been improved considerably in the past decade, but the general task can still not be considered as solved. It remains a lively research field with numerous methods and approaches of which only a few basic ones will be described below. A special case that will not be covered here is predominant melody extraction, a pitch detection task for which the input signal is polyphonic but only the salient (often vocal) melody pitches are extracted [47].

One of the first published *multi-pitch detection* systems was presented by Chafe et al. in 1985 [48]. They proposed to pick spectral magnitude peaks in a multiresolution Fourier transform (FT) and derive candidates for fundamental frequencies by grouping the peaks with respect to their frequency ratio. The detected candidates are then tracked over time in the spectrogram to discard spurious detections.

7.3.4.1 Iterative Subtraction

One class of multipitch detection systems is based on the principle of *iterative subtraction*. Here, a fundamental frequency detection algorithm for *monophonic* input signals is first applied to detect the predominant fundamental frequency. Second, this and related (mostly harmonic) frequency components are subtracted from the original signal. This process is then repeated on the residual signal until the criterion for termination (e.g. energy in the residual or maximum number of pitches) has been reached.

An early reference to such an algorithm in the spectral domain has been published by Parsons [49] who – inspired by the work of Schroeder [40] – constructs a histogram of spectral peaks and their integer submultiples, chooses the largest peak for the first fundamental frequency estimate, and removes this estimate and its multiples from the histogram to enable detection of a second fundamental frequency.

Klapuri et al. proposed two adaptations on the iterative subtraction procedure. They use an auditory-motivated monophonic fundamental frequency detection algorithm (see Section 7.3.3.8) to find the predominant, most salient fundamental frequency and estimate the spectrum of this pitch to subtract it from the original spectrum for the detection of additional pitches [50, 51].

De Cheveigné proposed a system for the tracking of multiple pitches in the time domain [52, 53]. First, the AMDF (see Section 7.3.3.3) is computed from the squared difference:

$$\mathrm{ASMDF}_{xx}(\eta, n) = \frac{1}{i_\mathrm{e}(n) - i_\mathrm{s}(n) + 1} \sum_{i=i_\mathrm{s}(n)}^{i_\mathrm{e}(n)} (x(i) - x(i+\eta))^2 \tag{7.33}$$

to detect the most salient period length η_{min} at the ASMDF minimum. Then, the detected frequency and its harmonics are removed from the signal by applying a comb cancellation filter to the signal with a delay corresponding to the lag of the detected minimum. The Finite Impulse Response (FIR) comb filter has the impulse response

$$h(i) = \delta(i) - \delta(i - \eta_{\mathrm{min}}) \tag{7.34}$$

and thus attenuates both the detected fundamental frequency and its harmonics at integer multiples. The remaining residual signal can then be used for the detection of a second fundamental frequency. It is also possible to implement this approach noniteratively by using an exhaustive search. In this case, two cascaded cancelation filters can be applied to the signal in all possible combinations of η_1 and η_2. The most likely pair of fundamental period lengths is the combination of η_1 and η_2 minimizing the overall output power of the output signals.

A similar idea has been presented by Miwa et al., who propose to use a cascade of comb filters, each comb filter subtracting a fundamental frequency with its harmonics from the input signal [54]. When the output of one filter approaches 0, it is clear that the signal contains the pitch at this filter frequency plus an unknown number of pitches represented by the preceding filters. Rerunning the analysis after shuffling the filter order then allows for detecting all pitches in the signal.

Meddis and Hewitt use an auditory approach similar to the one described in Section 7.3.3.8 for detecting one fundamental frequency and then use all remaining filter channels, i.e. filter channels not showing a peak at the first detected frequency, to detect additional fundamental frequencies [55]. The iteration process is terminated if more than 80% of the channels have been removed.

Similarly, Klapuri proposes to compute the normalized filterbank outputs after HWR [56]. He then computes the STFT of each filter band, sums their magnitudes

$$Z(k, n) = \sum_{c=0}^{C-1} |Z_c(k, n)|, \tag{7.35}$$

and weights the resulting overall spectrum $Z(k, n)$ with a low-pass filter transfer function. To identify possible fundamental frequency candidates, a set of delta pulse templates is used representing every detectable fundamental frequency with its harmonics. Similar to the calculation of the HSS, these templates are multiplied with the spectrum, and the result can be used as an estimate of the salience of each fundamental frequency. The components of the most salient frequency are then removed from the spectrum to find the next fundamental frequency in an iterative process.

Another auditory-inspired multi-pitch detection algorithm focusing on computational efficiency has been published by Karjalainen and Tolonen [57, 58]. They propose to use only two auditory bands with a cut-off frequency of 1 kHz and follow the general process described in Section 7.3.3.8 with HWR and smoothing. The approach has a few tweaks that are interesting to consider. First, they apply pre-whitening to flatten the spectral envelope in a preprocessing step. Second, they modify the autocorrelation function (ACF) calculation by setting $\beta = {}^2\!/_3$ in Eq. (16.38). Finally, the summary ACF is then harmonically processed with an idea similar to HPS. This harmonic postprocessing is an iterative process in which an interpolated, time-scaled, and half-wave rectified version is subtracted from the half-wave rectified ACF $r(\eta, n) = r(j, \eta, n)$ to eliminate peaks at multiples of the fundamental frequency. The repeated application of HWR in combination with scaling aims at discarding frequencies other than the fundamental frequencies so that peaks only remain at actual fundamental frequencies and not their multiples.

7.3.4.2 Nonnegative Matrix Factorization

Non-negative Matrix Factorization (NMF), introduced by Lee and Seung [59], has attracted considerable attention in the context of multi-pitch detection during the last decade. It decomposes a time-frequency representation into a dictionary matrix containing the spectra of the individual sounds and another matrix containing the information on when each of the individual spectra is active.

NMF is an iterative algorithm which estimates an approximation $\hat{\boldsymbol{X}}$ of a given matrix $\boldsymbol{X}$ through a multiplication of the iteratively updated matrices $\boldsymbol{W}$ and $\boldsymbol{H}$:

$$\hat{\boldsymbol{X}} \approx \boldsymbol{W} \cdot \boldsymbol{H}. \tag{7.36}$$

If the dimension of both $\boldsymbol{X}$ and $\hat{\boldsymbol{X}}$ is $k \times n$, then the dimensions of $\boldsymbol{W}$ and $\boldsymbol{H}$ will be $k \times r$ and $r \times n$, respectively, with r indicating the NMF rank. No matrix can contain negative entries. Alternatively, the problem can formulated as [60]

$$\boldsymbol{X} = \sum_{i=0}^{r-1} w_i \cdot h_i + \boldsymbol{E}, \tag{7.37}$$

with $\boldsymbol{X} \in \mathbb{R}^{k\times n}$, $\boldsymbol{W} = [w_0, w_1, \ldots, w_{r-1}] \in \mathbb{R}^{k\times r}$, and $\boldsymbol{H} = [h_0, h_1, \ldots, h_{r-1}]^T \in \mathbb{R}^{r\times n}$. Figure 7.22 visualizes these dimensions. Once the elements of the error matrix $\boldsymbol{E}$ are minimized, the matrix $\boldsymbol{X}$ can be represented by the combination of the *dictionary* matrix $\boldsymbol{W}$ and the *activation* matrix $\boldsymbol{H}$. The goal of the optimization algorithm is to iteratively minimize the error matrix $\boldsymbol{E} = \boldsymbol{X} - \hat{\boldsymbol{X}}$, which is the distance of the

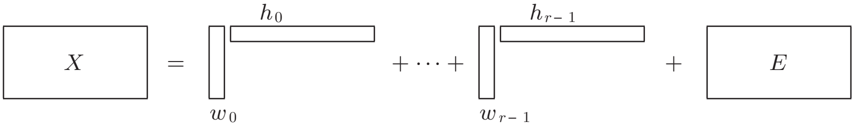


Figure 7.22 Visualization of Eq. (7.37) with example dimensions.

original matrix $\boldsymbol{X}$ and the estimated matrix $\hat{\boldsymbol{X}} = \boldsymbol{W} \cdot \boldsymbol{H}$. The two most common loss functions are the following: the squared Euclidean distance

$$d_{\mathrm{EU}}(\boldsymbol{X}, \hat{\boldsymbol{X}}) = \left\| \boldsymbol{X} - \hat{\boldsymbol{X}} \right\|_2^2 = \sum_{kn} (x_{kn} - \hat{x}_{kn})^2, \text{ and} \tag{7.38}$$

the generalized Kullback–Leibler divergence

$$d_{\mathrm{KL}}(\boldsymbol{X}, \hat{\boldsymbol{X}}) = \sum_{kn} x_{kn} \log \left(\frac{x_{kn}}{\hat{x}_{kn}} \right) - x_{kn} + \hat{x}_{kn}. \tag{7.39}$$

Also compare Eqs. (4.5) and (4.9).

Gradient descent can be used to minimize the distance; in practice, the matrices $\boldsymbol{W}$ and $\boldsymbol{H}$ can be updated alternating with multiplicative update rules [59]:

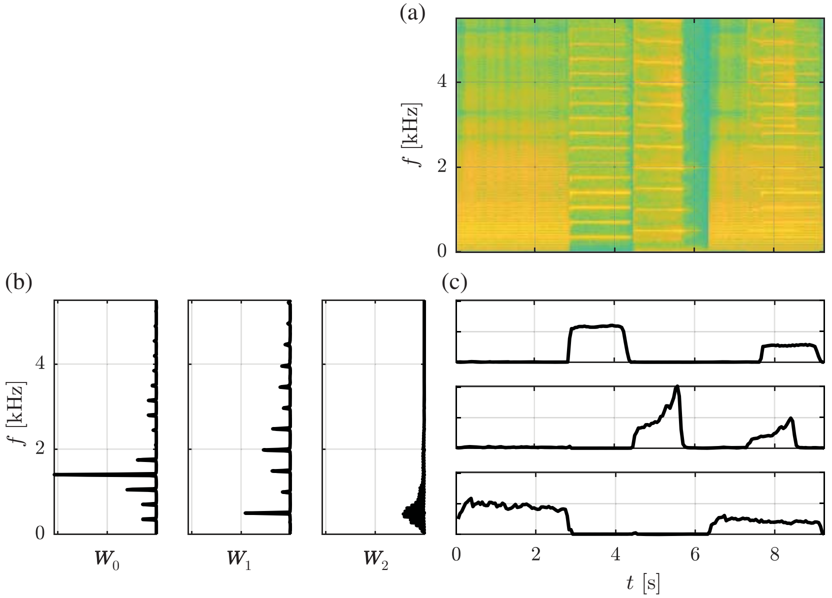
$$H \leftarrow H \frac{(W^T X)}{(W^T W H)} \tag{7.40}$$

$$W \leftarrow W \frac{(V H^T)}{(W H H^T)} \tag{7.41}$$

Given the introduction of NMF above, the question remains what practical use this algorithm has for pitch tracking. Smaragdis and Brown answered this question by applying NMF to the approximation of the nonnegative magnitude spectrogram [61]. Under the assumption that individual sound components such as pitches simply add up in the magnitude spectrum, the magnitude spectrum is decomposed in r spectral templates of the sound components in the dictionary matrix $\boldsymbol{W}$ and their salience over blocks in the activation matrix $\boldsymbol{H}$.

Figure 7.23 shows an example of a factorization. The input signal is constructed from three signals: the pitch D♭2 played by a horn, the pitch F4 played by an oboe, and the pitch B4 played by a violin. First, the three signals are concatenated (horn 0–3 s, oboe 3–4 s, violin 4–6 s), then mixed to appear simultaneously at a lower level (6–9 s). If we look at the automatically estimated dictionary matrix, we see that the algorithm successfully separated the three pitches into the individual templates w_0: oboe, w_1: violin, and w_2: horn (note that the order of the templates is random and might change if run again). The activation matrix on the right indicates clearly the contribution of each template to the mixture at each point in time.

Thus, NMF is capable of separating different pitches in a polyphonic mixture. However, to apply this approach to pitch tracking, each dictionary template has to be labeled retroactively with the corresponding pitch, as the templates are unordered and the pitch content might not be known beforehand. If the factorization worked as intended, this should be an easy task as each template contains only one harmonic series. It is also possible to assign the pitches only once instead of doing that after each factorization; in this case, the dictionary matrix is kept fixed after the assignment and the factorization of new signals only



Repo 1: plotF0Nmf.m

Figure 7.23 Example for the utilization of NMF for detecting the individual pitches: input spectrogram (a), estimated template vectors (b), and estimated activation vectors (c) for a signal containing three pitches played individually first and simultaneously towards the end of the file.

updates the activation matrix. Each row of the resulting matrix will then indicate the salience of one pitch over time.

Listing 7.7 shows the implementation of the iteration loop of NMF.

The algorithmic simplicity and efficiency of the NMF approach make it an appealing method for fundamental pitch extraction. In theory, the superposition of sounds is not purely additive due to interference, although in practice that rarely has significant impact on the results. There are a number of practical challenges, however, that potentially impact the usability and robustness of the default algorithm. First, there is an obvious tradeoff between frequency resolution and time resolution (see Section 7.3.1.2) that usually leads to insufficient resolution for low-frequency signals. Second, the rank r is a hyperparameter that should preferably be adjusted for each individual input signal depending on the pitch content of the signal. Choosing a high rank will likely lead to multiple templates per pitch, making the pitch assignment hard. Choosing a low rank will negatively impact how well the spectrogram can be reconstructed and will lead to one template representing multiple pitches. Third, there are not only periodic, pitched components in a real-world audio signal. Percussive, noise-like components have to be modeled as well in some form, requiring additional unpitched templates

```
    % iteration
    while (count < iMaxIteration)

        % current estimate
        X_hat = W * H;

        % update
        if bUpdateH
            H = H .* (W'* (X./X_hat)) ./ (W'*rep);
        end
        if bUpdateW
            W = W .* ((X./X_hat)*H') ./ (rep*H');
        end

        %normalize
        for i = 1:iRank
            W(:, i) = W(:, i) ./ (norm(W(:, i), 1));
        end

        %calculate variation between iterations
        count = count + 1;
        err(count) = KlDivergence_I(X, (W*H)) + fSparsity * norm(H, 1);

        if (count >=2)
            if (abs(err(count) - err(count - 1)) / ...
                    (err(1) - err(count) + realmin)) < 0.001
                break;
            end
        end
    end
```

Repo 2: ToolSimpleNmf.m
Repo 3: ToolSimpleNmf.py

Listing 7.7 Example implementation of NMF.

(which might, on the other hand, help in detecting or separating drums). Fourth, differences in intonation cannot be easily modeled, as the template only contains one harmonic series per pitch. Transposing an input signal by, e.g. a quartertone will result in different templates. Playing techniques such as vibrato cannot be properly modeled due to this pitch quantization. Fifth, multitimbral sources with multiple instruments can be hard to model. If two instruments distinct in timbre play the same pitch, the template for this pitch will probably converge to some compromise spectral shape or we end up with two highly correlated templates. Sixth, the basic NMF approach assumes that the spectral shape of a template is time-invariant and only scaled up and down by the activation; therefore, it cannot model component spectra that change over time. Thus, instrument sounds with timbre changes within one note are difficult to model. Seventh, when adapting the templates to real-world music data, an imbalanced pitch distribution in the data might lead to inaccurate or even missing templates for infrequently occurring pitches as the algorithm only minimizes the global, overall error.

After the introduction of NMF to the task of polyphonic pitch tracking [61], many publications followed refining the pitch-tracking process with NMF. One very common modification of the standard algorithm in the context of pitch tracking is the use of pretrained templates with each template corresponding to one specific pitch. This makes postanalysis for the pitch-mapping of each template unnecessary because each template is already associated with a specific pitch. During inference, the template matrix is kept fixed and does not have to be updated, only the activation matrix is iteratively updated [62, 63]. This is a particularly useful modification for the analysis of instruments with stable timbre and quantized pitches such as the piano.

Many other improvements to the NMF approach for multipitch tracking have been proposed over time. Replacing the loss function, for example, can improve the results for specific use cases [63, 64]. Other researchers proposed modifying established loss functions; Cont applied a sparsity constraint on the activation matrix to ensure a minimum of active templates at each time [62]. The factorization approach can also be modified on a more fundamental level by utilizing probabilistic approaches [65–67] and introducing shift-invariance [65].

NMF has also been applied to various related and unrelated other tasks in Audio Content Analysis (ACA). The most prominent example is probably source separation: if individual templates can be assigned to specific instruments and/or speakers, approximating the magnitude spectrogram of only one source is possible by setting the activations of all other sources to zero. This source signal can then be reconstructed by using the original phase for reconstructing the time domain signal from the masked magnitude spectrogram [68–71]. The method has also been utilized for instrument recognition in general [72, 73] and for the transcription for nonpitched drum sounds [74, 75]. A more detailed overview of instrument recognition and drum transcription can be found in Sections 14 and 9.8, respectively. Other application examples range from identifying structural components in music [76–78], recognizing mixed sound snippets [79], and denoising [64].

7.3.4.3 Other Approaches

Probabilistic approaches aim at modeling the pitch-tracking problem by means of a statistical framework. To give only one example, Kameoka et al. model a tone in the frequency domain as a superposition of weighted Gaussian distributions at integer multiples of the fundamental frequency and its power envelope function in the time domain by overlapping Gaussian distributions [80]. The spectrogram is then decomposed into clusters which model individual notes.

While the majority of state-of-the-art (multi-)pitch detection systems are now based on neural networks, the adaption of deep neural networks (DNNs) has been somewhat slower than for other tasks. Marolt successfully introduced neural networks to the task of piano transcription in the early 2000s [81]. Later, various input representations and network architectures have been proposed (e.g. [82–85]). Many of these methods focus exclusively on piano music and progress seems to be slowed by the lack of accurately annotated training data [86].

7.3.5 Evaluation

The evaluation of pitch-tracking systems appears to be easy enough: predict the pitch and verify it against the ground truth data. A closer look at the ground truth of some available datasets, however, reveals a more nuanced picture. This is partly due to specific task constraints; the available datasets can be roughly categorized into classes of increasing task complexity: (i) the detection of the pitch of single instrument notes (one note), (ii) the detection of the pitches of a monophonic melody (series of nonoverlapping notes), (iii) the detection of the pitches of a predominant melody in a polyphonic mixture of instruments (series of nonoverlapping notes), and (iv) the detection of all pitches in a polyphonic mixture of instruments (series of overlapping notes).

Similarly, the datasets have to vary depending on whether the goal is to extract the fundamental frequency trajectory or to transcribe to a score-like result. For instance, comparing two pitch-tracking datasets, we find that the TONAS dataset [87] provides an fundamental frequency value in Hertz every 512 audio samples, while MusicNet provides the start and stop times in seconds of individual MIDI pitches [88]. These two examples showcase different views of how the goals of "pitch tracking" as a task can vary and how pitch-tracking systems can be evaluated. First, they differ in frequency granularity: MusicNet targets reconstructing a musical score and thus needs only one value per musical pitch, while TONAS provides a "continuous" floating point value for the fundamental frequency, enabling the analysis of intonation and pitch deviations. Second, the MusicNet annotations are segmented by note start and stop times while TONAS provides equidistant annotations with comparably high time resolution. Thus, although both datasets have been designed for the seemingly identical task of pitch tracking, they differ considerably in their ground truth annotations. Inquiries such as the analysis of intonation, time-variant pitch deviations, and performance techniques such as vibrato, for example are only possible with high granularity. Thus, not only could it be argued that they represent (slightly) different tasks, but they also require different evaluation approaches and metrics.

7.3.5.1 Metrics

The nonlinearity of human pitch perception needs to be reflected by the evaluation. Therefore, all metrics should be computed in the pitch domain rather than the frequency domain to ensure that the computed evaluation results are at least somewhat meaningful from a perceptual point of view. It is common to run the evaluation in the MIDI pitch domain as introduced in Eq. (7.11).

If the ground truth provides only (quantized) MIDI pitches, the evaluation simplifies to a classification task in which each pitch is either correct or incorrect. In that case, the detected fundamental frequencies are mapped to the closest equal-tempered pitch and classification metrics can be reported. Typical classification-based metrics are the accuracy and F-Measure, which are based on the number of correct and incorrect predictions (see Section 6.1.1). Given a set of $\mathcal{N}$ ground truth pitch labels $\mathfrak{p}_{\mathrm{GT}}$ and the corresponding estimated pitches $\hat{\mathfrak{p}}$, Salamon et al. [89] summarized a set of commonly used metrics as raw pitch accuracy

$$\mathrm{RPA} = \frac{\sum\limits_{\forall n} \mathrm{TP}_n}{\mathcal{N}} \tag{7.42}$$

with

$$\mathrm{TP}_n = \begin{cases} 0, & \text{if } |\mathfrak{p}_{\mathrm{GT}}(n) - \hat{\mathfrak{p}}(n)| \geq 0.5 \\ 1, & \text{otherwise,} \end{cases}$$

and raw chroma accuracy counting octave errors as match

$$\mathrm{RCA} = \frac{\sum\limits_{\forall n} \mathrm{TP}_{\mathrm{chroma},n}}{\mathcal{N}} \tag{7.43}$$

with

$$\mathrm{TP}_{\mathrm{chroma},n} = \begin{cases} 0, & \text{if mod } \big(|\mathfrak{p}_{\mathrm{GT}}(n) - \hat{\mathfrak{p}}(n)|, 12\big) \geq 0.5 \\ 1, & \text{otherwise} \end{cases} .$$

In addition, they propose an overall accuracy metric combining the pitch detection accuracy with voicing detection accuracy. Voicing (voice present or not), also referred to as *(Vocal) Activity*, is an additional, not directly pitch-related, variable to be potentially evaluated. Similarly, it has been argued that even metrical structure could be part of a pitch detection evaluation [90]. There are other metrics that have been proposed for multipitch detection systems, including substitution error, miss and false alarm errors, and an overall error score [91].

If the ground truth provides fundamental frequencies (or more precisely, floating point resolution pitch values), the classification metrics above can only be computed by assuming a detection tolerance (e.g. ±50 Cent in the example above). Deviation-based metrics, however, may provide more insights into the algorithm accuracy. A typical standard metric is, for example, the mean average error (MAE) (see Section 6.1.2). Other metrics, such as the MSE or the standard deviation from the ground truth are also potential measures for accuracy. Bittner and Bosch proposed to generalize the binary metrics above to deviation-based metrics [92].

The metrics mentioned above are all *frame-level* metrics, i.e. they are based on each ground truth value representing a short span of time. The time resolution of the ground truth will vary between datasets and is unlikely to match the time resolution of the system to be evaluated. Unfortunately, there is no obvious solution to this problem beyond changing the system's output time resolution; the interpolation of the predicted values is probably the best way to address this mismatch. Any nontrivial interpolation that does not violate the sampling theorem should be applicable. Obviously, the ground truth needs to remain unchanged.

7.3.5.2 Datasets

Although pitch transcription is a core task in the analysis of musical audio, the amount, size, and breadth of available annotated data do not necessarily reflect this importance. The issue of annotation, requiring considerable effort from experts, means that dataset annotations are either automatically generated (e.g. MusicNet [88]), created through instrument sensors (e.g. MAESTRO [93]) or limited (e.g. [94]). Furthermore, there are only few datasets beyond piano and singing (e.g. Guitarset [95]). In the past, this lack of both data and diversity has impeded the progress of sophisticated methods for pitch transcription.

7.3.5.3 Results

Given the data limitations mentioned above, any evaluation results on these datasets will only have limited meaning. However, the existing results can still give us an impression of the general capabilities of modern systems. In a recent evaluation, most systems yielded overall accuracies between 0.5 and 0.7 on one dataset and accuracies from 0.2 to 0.5 on a different dataset.[5] While these data most decidedly contain challenging, multi-instrument recordings, it seems safe to say that current state-of-the-art systems do not appear to be sufficiently reliable for general music transcription.

7.4 Tuning Frequency Estimation

The *tuning frequency* f_{A4} is the frequency of the *concert pitch* $A4$ (also: *standard pitch*) which is used for tuning one or more musical instruments. The tuning frequency is standardized internationally to 440 Hz [96], but the exact frequency used by musicians can vary due to various reasons such as the use of historic instruments or timbre preferences. Two performances of the same piece of music using different tuning frequencies will differ in their average pitch height, but will not change the frequency ratios between pitches like different temperaments.

The range of typical tuning frequencies decreased over the centuries. Table 7.5 shows this range for the past three centuries as deviation from 440 Hz [24].

---

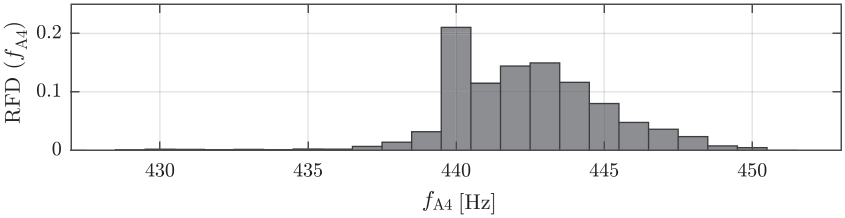
[5] MIREX 2019. https://www.music-ir.org/mirex/wiki/2019:MIREX2019_Results. Last retrieved on 4 July, 2021.

Table 7.5 Typical range of deviation of the tuning frequency from 440 Hz over three centuries.

Year	Lower Deviation (Hz)	Upper Deviation (Hz)
1750	−50	+30
1850	−20	+20
1950	−5	+10

Nowadays, while for many electronic music productions, the "standard" tuning frequency of 440 Hz is used, the tuning frequencies of orchestras still deviate from this standard tuning frequency. For example, the Chicago Symphony Orchestra and the New York Philharmonic tune at 442 Hz, while the Berliner Philharmoniker and the Wiener Philharmoniker have a tuning frequency of 443 Hz.[6] At least in the case of both European orchestras, the tuning frequency used to be higher in previous decades. The frequencies 442 and 443 Hz correspond to deviations of 7.85 and 11.76 Cent from the standard tuning frequency, respectively. In addition to musicians consciously choosing a tuning frequency, historic recordings might also be reproduced at nonstandard tuning frequencies because of speed deviations between recording and playback equipment.

There does not exist much data on the actual tuning frequency of real-world recordings. Zhu et al. processed a database of 60 popular and 12 classical pieces[7] and found only three pieces of this database with a deviation of approximately 2–4 Cent from the standard tuning frequency [97]. Lerch investigated the tuning frequencies in a proprietary dataset of classical music, consisting of more than 3000 tracks and an overall playing time of approximately 291 hours [98]. The result is shown in Figure 7.24: the distribution has a maximum at a tuning frequency of 440 Hz; the maximum itself consists of about 21% of the dataset.



Repo 1: plotTuningFreqs.m

Figure 7.24 Distribution of tuning frequencies. Source: Data from Lerch [98]). .

---

[6] According to the orchestra's archivists, March and April 2006.

[7] The term *classical music* is in this context understood as "nonpopular" music (that is generally not beat-based), as opposed to the epoch itself.

The arithmetic mean of the distribution, however, is at 442.38 Hz (its standard deviation is 2.75 Hz) as the results are clearly skewed toward higher tuning frequencies: the majority of the results (95%) is in the range from 439 to 448 Hz, while the percentage of files below 439 Hz is about 3.3%. Note that only 50% of the results have a tuning frequency in the range of 440–443 Hz.

While the studies above assume a static tuning frequency once the instruments have been tuned, this assumption is not necessarily true. The tuning frequency may change over the course of a performance or a recording session. On the one hand, the tuning frequency could be slowly decreasing as it sometimes happens at *a cappella* performances, on the other hand, the tuning frequency may slightly increase, for example due to a rising involvement of the musicians during the concert. The maximum range of this deviation, however, can be assumed to be small in the case of professional musicians (about 3–5 Cent).

Most pitch-based ACA tasks utilize the tuning frequency, however, many systems build on the assumption of a fixed tuning frequency at 440 Hz. While this assumption might work reasonably well in some cases, using a correct estimate of the tuning frequency should generally improve detection accuracy [98]. The estimation of the tuning frequency f_{A4} is a prerequisite for every mapping from frequency to musical pitch given in Eq. (7.11). Examples of tasks utilizing this mapping are key detection, chord recognition, and pitch tracking. For most nonpitch-related tasks, the tuning frequency is irrelevant, although it has been identified as a potentially confounding variable being learned from data [99].

7.4.1 Approaches to Tuning Frequency Estimation

A standard approach to *tuning frequency estimation* has the following processing steps: (i) a time-frequency transformation to extract the frequencies of the tonal frequency components with a high-frequency resolution, (ii) the computation of a deviation of the extracted frequencies from an equal-tempered pitch grid based on the assumed tuning frequency, and (iii) an optimization process minimizing the average deviation by adjusting the tuning frequency estimate. To give a specific example, an early system for tuning frequency estimation using a bank of steep resonance filters with adjustable midfrequencies is presented below [98, 100]. Each (equal tempered) semitone in the range of two octaves is represented by a group of narrowly spaced filters, with each group centered around the semitone frequency based on the most recent tuning frequency estimate. Under the assumption that the filter output energies of the filters left of the semitone frequency should equal the output energies right to the semitone frequency in the case of a correct tuning frequency estimate, the symmetry or balance of the sum of the output energies of all “left” filters compare to the sum of all “right” filters is evaluated as model error or objective function. If the energies are unequal, the tuning frequency, and thus all filter midfrequencies, is adapted with the goal of making the energy distribution more symmetrical in the future. The adaption rule for the adjustment of the estimated tuning frequency $\hat{f}_{A4}$ of the next processing block $n + 1$ does not require a signal-dependent adaption step size [101], but uses only the imbalance direction:

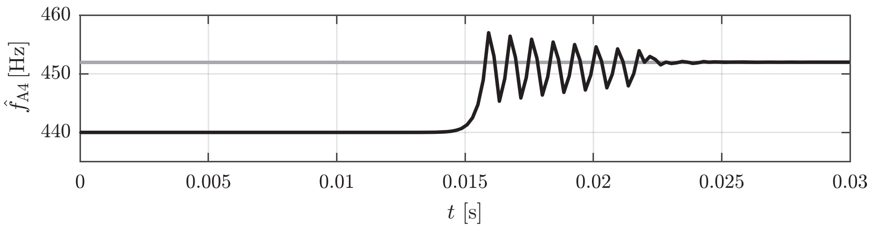
$$\hat{f}_{A4}(n+1) = \big(1 + \mathrm{sign}\,(E(2) - E(0)) \cdot \lambda\big) \cdot \hat{f}_{A4}(n) \tag{7.44}$$

with η being scaled up if the direction of the previous adaptation was the same and scaled down otherwise. The advantage of the scaling is an increasing step size as long as the sign does not change and a decreasing step size otherwise. Figure 7.25 shows the adaptation from the initial tuning frequency of 440 Hz to the real frequency of 452 Hz. Adaptation can be tuned either for speed or for accuracy. The advantage of constant adaption is (i) applicability to real-time systems, and (ii) capabilities to track a slowly changing tuning frequency over time.

The assumption of equal temperament is common but not necessarily true; only a score-informed approach with knowledge of the used temperament (or the automatic detection of key and temperament) would render this assumption unnecessary. One such score-based approach was proposed by Scheirer, who swept a set of narrow band-pass filters over a small frequency range around handpicked frequencies to match pitches from the previously analyzed score to identify the frequency maximizing the output energy [102].

If an STFT is used for the frequency representation, it is common to either compute the instantaneous frequency of the detected peaks or use some other form of spectral interpolation to ensure sufficient frequency accuracy [103, 104]. Alternatively, a high-resolution constant Q transform (CQT) is a good choice to yield a constant pitch resolution (as opposed to constant frequency resolution) [97, 105].

The objective function for estimating the tuning frequency is commonly based on energy per deviation. One way of doing so is to create a histogram-like representation of the deviations from the closest equal tempered pitch. The location of the maximum [97, 106] or the location of the "histogram mass center" [107] can then be chosen to derive the estimated tuning frequency. The deviation can also be formulated in circular statistics as proposed by Dressler and Streich [108].



Repo 1: plotfA4Rprop.m

Figure 7.25 Adaptation of the tuning frequency estimate from an initial setting of 440 Hz to the target frequency of 452 Hz with the RPROP algorithm.

7.4.2 Evaluation

The creation of a dataset with real-world audio data and reliable ground truth tuning frequency annotations is nearly impossible, and no such dataset exists [109]. To complicate the evaluation further, even when tuning frequency data are known, such as in the case of some orchestras, there is no guarantee that the available recordings always match that tuning frequency. For instance, historic recordings can suffer from tape speed mismatches between recording and reproduction, resulting in audio tuned slightly higher or slightly lower. Furthermore, the annotation of existing recordings with their tuning frequencies requires experts with good ears and is so tiring that it is hardly doable. The only possible solution to this dilemma is the use of synthesized audio as digital oscillators and/or sound libraries can be adjusted in their tuning. This has the inherent disadvantage that the differences between synthesized audio and recorded audio might lead to overestimation of the reliability of tuning frequency algorithms in real-world scenarios.

As tuning frequency estimation is a regression task, deviation-based metrics can be used, although metrics for classification can also be reported when assuming a tolerance. Note that the evaluation domain should be the unit Cent to account for (i) the nonlinearity of frequency perception and (ii) the pitch-independence of tuning frequency.

7.5 Key Detection

The musical *key* of a tonal piece of music is defined by both its *mode* and its *tonic*. The *tonic* is the most important pitch class in a specific key. It is also referred to as the *first scale degree* and is often the most common pitch class in the piece. The *mode* defines a set of relative pitch relationships, such as the distance between first and second scale degree is a major second, between first and third scale degree is a major third, etc. The most common modes are the *major mode* and the *minor mode*. Figure 7.26 displays an example set of different modes, all starting from the tonic *C*. All modes except the major mode and the two minor modes – the aeolian mode and the harmonic mode – are only of interest in specific musical styles. Therefore, researchers in the field of ACA often ignore these other modes. The key thus defines the set of pitch classes used to construct the tonal aspects of a piece of music. In popular music, it is most likely for a piece to have exactly one key. There are many exceptions to this rule: the key can change within a piece (when a so-called *modulation* occurs) and non-key pitch classes may be used for musical reasons.

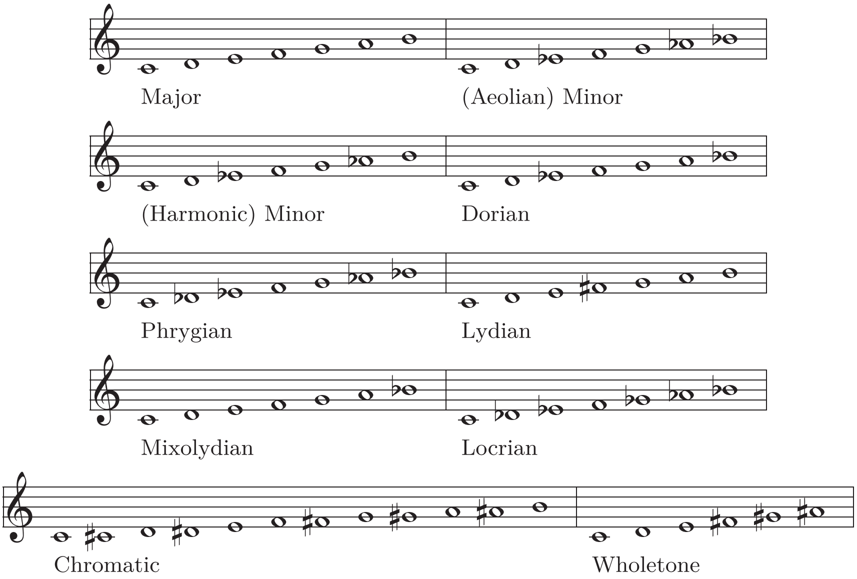


Figure 7.26 Different modes in musical score notation starting at the tonic *C*.

Depending on the tonic and mode, up to six different accidentals[8] have to be used to raise or lower the pitches in the musical score. The notational convention allows writing all key-inherent accidentals at the begin of a staff – the *key signature* – and to add only accidentals within the score where nonkey-inherent pitches are used.

Figure 7.27 shows major modes with their key signatures starting from all possible tonics. As can be seen from the score, the tonics of keys that differ only in one accidental are always spaced by a fifth (*F* ↔ *C*, *C* ↔ *G*, *G* ↔ *D*, etc.). This relationship can be visualized by the so-called *circle of fifths* (Figure 7.28).[9] Neighboring keys on the circle of fifths have all pitch classes but one in common; for example *G Major* has the same pitches as *C Major* except for the *F* which is raised to an *F*♯. The circle of fifths can thus be understood as a model for a distance map between keys.

The circle of fifths also exists for the (aeolian) minor keys with *A Minor* being the key without accidentals. Keys construed from the same set of pitch classes such as *C Major* and *A Minor* are called relative keys; in the circle of fifths their

---

[8] Theoretically, any number of accidentals is possible, but assuming enharmonic equivalence (see below), six accidentals cover all possibilities.

[9] Strictly speaking, the circle of fifths is only closed in the case of enharmonic equivalence when, e.g. *F*♯ equals *G*♭ (see Section 7.2.3.1).

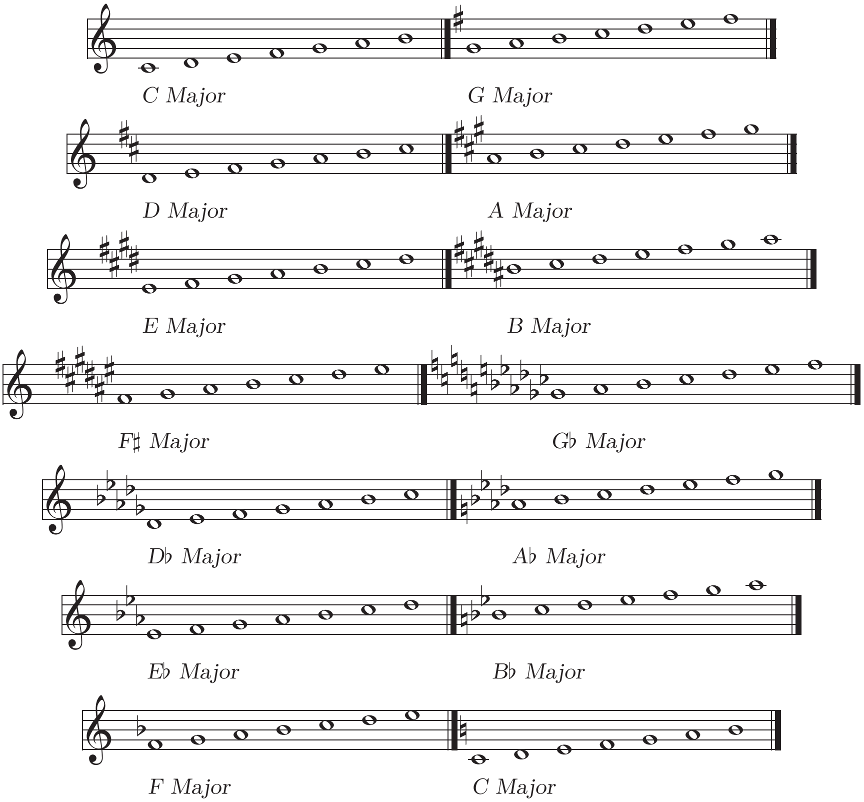


Figure 7.27 The twelve major scales in musical score notation, notated in the 4th octave. Note that *C Major* is repeated to indicate the circularity of the circle of fifths.

distance is 0. In order to build an analytical model for key distances with a nonzero distance between relative keys, the circle of fifths can be enhanced to a three-dimensional model. This model would feature two parallel planes, one containing the circle for major keys and the other for minor keys. Parallel keys thus share the same (x, y) coordinates but have a different z coordinate (which is then the distance between the two parallel keys).

Since the circle of fifths shows the relation of different keys, it hints also at what modulations are more or less likely. To give an example, the most likely key changes from *F Major* would be either *C Major*, *B♭ Major*, or *D Minor*.

The musical key is an important tonal property of a piece of tonal music as it signifies the tonal center and restricts the pitch classes used within this tonal context. In classical music, the key is used as a standard descriptor used to identify a specific piece of music, complementing name, genre/instrumentation, and opus. In modern software applications for DJs, the key can be used to display the

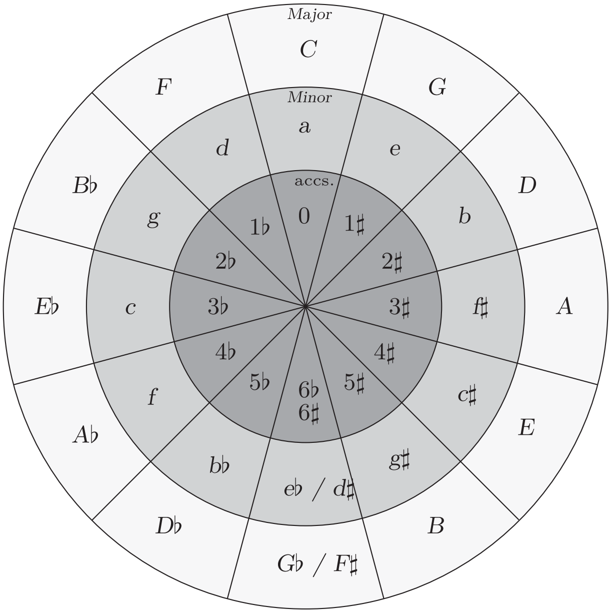


Figure 7.28 Circle of fifths for both major keys and minor keys, plus the number of accidentals of the key signature per key.

tonal compatibility between two tracks, i.e. to visualize how much "tonal overlap" they have to aid so-called harmonic mixing for the creation of mash-ups.

Since the key can be seen as a set of "allowed" pitch classes, the use of an octave-independent representation of pitch is sufficient for its automatic detection. The most widely used representation is the so-called pitch chroma, which is introduced in the following.

7.5.1 Pitch Chroma

The *pitch chroma* or *pitch class profile* is a histogram-like 12-dimensional vector with each dimension representing one pitch class (C, $C\sharp$, D, ..., B; see Section 7.2.1). It can be seen as a *pitch class distribution* for which the value of each dimension may represent both the number of occurrences of the specific pitch class in a timeframe and its energy or velocity throughout the analysis block.

The pitch chroma is probably the most common instantaneous feature used to describe the tonal content of audio. Similar to the timbre descriptors introduced in Section 3.5, it is computed from a spectrum (normally from a STFT). As mentioned above, it is a multidimensional feature with 12 pitch classes. There are several advantages of pitch chroma-based analysis. It is less dependent on timbre fluctuations and noise than other features describing tonal content. The pitch chroma is also robust against loudness fluctuations (if normalized accordingly) as well as against octave errors – a typical problem of pitch detection algorithms – with the self-evident disadvantage that all octave information is lost. There is, for example, no distinction in the pitch chroma representation of a held note, a note repetition, and an octave interval.

While a representation of pitch class similar to the pitch chroma has been frequently used in the past (see, e.g. Krumhansl's tonal distributions [110] and compare Section 7.1.2), Fujishima might have been the first to propose its use in the context of audio signal processing [111]. The pitch chroma representation then quickly became popular [112, 113] and continues to be frequently referred to in ACA publications.

The exact algorithmic description of the pitch chroma computation varies from implementation to implementation; in all cases

- a frequency representation of the audio signal block is grouped into semitone bands,
- a measure of salience is computed for each band, and finally,
- the sum of all bands (over all octaves) corresponding to a specific pitch class is calculated.

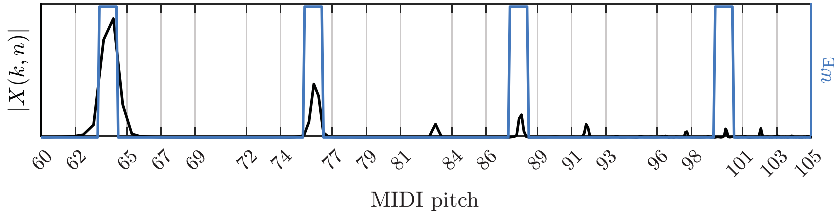
The simplest way to extract the pitch chroma sums the squared STFT magnitudes in each semitone band within the boundary indices $k_\mathrm{l}, k_\mathrm{u}$, and the result in every octave o is added to the corresponding pitch chroma entry with pitch class index j:

$$v(j,n) = \sum_{o=o_\mathrm{l}}^{o_\mathrm{u}} \left(\frac{1}{k_\mathrm{u}(o,j) - k_\mathrm{l}(o,j) + 1} \sum_{k=k_\mathrm{l}(o,j)}^{k_\mathrm{u}(o,j)} |X(k,n)|^2 \right), \tag{7.45}$$

$$v(n) = [v(0,n),\ v(1,n),\ v(2,n),\ \dots,\ v(10,n),\ v(11,n)]^\mathrm{T}. \tag{7.46}$$

The indices $k_\mathrm{l}, k_\mathrm{u}$ are located at a distance of 50 Cent from the midfrequency of each equal-tempered pitch.

Figure 7.29 shows the semitone bands for the pitch class E (typically pitch class index 4) in the octaves 5–8. Note that the window bandwidth appears constant



Repo 1: plotPitchChromaGrouping.m

Figure 7.29 Mask function for pitch class E for pitch chroma computation (octaves 5–8).

on the logarithmic pitch axis, while on the linear frequency axis it would increase with increasing frequency.

Frequently, the pitch chroma is normed so that the sum of all possible pitch classes equals 1:

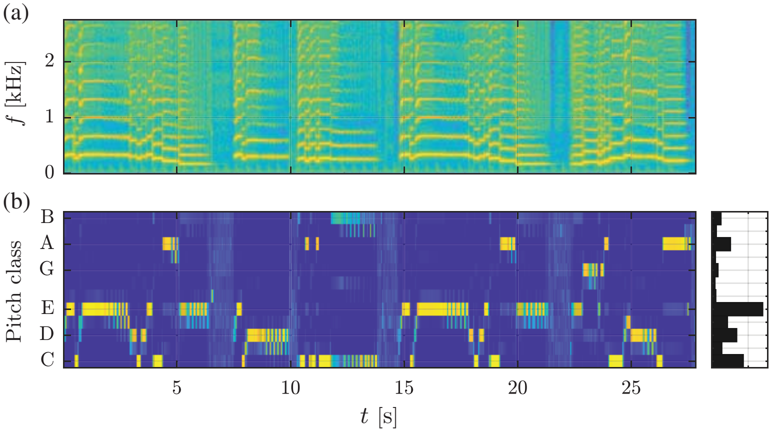
$$v_{\mathrm{N}}(n) = v(n) \cdot \frac{1}{\sum\limits_{j=0}^{11} v(j,n)}. \tag{7.47}$$

Alternatively, the pitch chroma can be interpreted as a vector and is normed to a length of 1 instead:

$$v_{\mathrm{N}}(n) = v(n) \cdot \sqrt{\frac{1}{\sum\limits_{j=0}^{11} v(j,n)^2}}. \tag{7.48}$$

However, it should be kept in mind for normalization that the pitch chroma is neither a series of observations nor a position in a space with 12 unrelated dimensions; it is a *distribution*. It should also be noted that by normalizing each pitch chroma per block, any loudness or intensity information is removed.

Figure 7.30 shows the magnitude spectrogram and the pitch chromagram of a monophonic audio file. It allows the identification of individual pitch classes and thus an octave-independent reconstruction of the melody. The aggregated average pitch chroma on the bottom right shows which pitches are more or less dominant over the whole recording. The input audio is the same saxophone recording used to generate Figure 3.11.



Repo 1: plotPitchChroma.m

Figure 7.30 Magnitude spectrogram (a) and the pitch chromagram (b) of a monophonic saxophone recording. The bottom right shows the aggregated average pitch chroma for the whole recording.

The pitch chromagram can be extracted from the spectrogram with a simple matrix multiplication; the matrix contains the mask function for each pitch and has a dimension of $12 \times \mathcal{K}$. Listing 7.8 shows the code for this matrix multiplication. The generation of the mask matrix can be found in the source code available through the link.

Note the construction of the pitch chroma filters requires a tuning frequency; this means that this pitch chroma simply assuming a standard tuning frequency will be less usable with detuned audio inputs.

```
    % generate filter matrix
    H = GeneratePcFilters_I(size(X, 1), f_s);

    % compute pitch chroma
    vpc = H * X.^2;

    % norm pitch chroma to a sum of 1
    vpc = vpc ./ repmat(sum(vpc, 1), 12, 1);

    % avoid NaN for silence frames
    vpc (:,sum(X, 1) == 0) = 0;
end

%> generate the semi-tone filters (simple averaging)
function [H] = GeneratePcFilters_I (iSpecLength, f_s)

    % initialization at C4
    f_mid = 261.63;
    iNumOctaves = 4;

    % sanity check
    while (f_mid*2^iNumOctaves > f_s/2 )
        iNumOctaves = iNumOctaves - 1;
    end

    H = zeros (12, iSpecLength);

    for i = 1:12
        % semitone boundaries in Hz
        afBounds  = [2^(-1/24) 2^(1/24)] * f_mid * 2 * (iSpecLength-1) / f_s;
        for j = 1:iNumOctaves
           iBounds =    [ceil(2^(j-1)*afBounds(1)) ...
                        floor(2^(j-1)*afBounds(2))]+1;
           H(i, iBounds(1):iBounds(2)) = 1 / (iBounds(2)+1-iBounds(1));
        end

        % increment to next semi-tone
        f_mid = f_mid * 2^(1/12);
    end
end
```

Repo 2: FeatureSpectralPitchChroma.m
Repo 3: FeatureSpectralPitchChroma.py

Listing 7.8 Computation of the pitch chroma with a matrix multiplication.

As with many other common features, implementations of the pitch chroma can significantly differ. This starts with different parametrizations of the basic pitch chroma mentioned above: different STFT block and hop lengths, a different number of octaves to integrate, or using the magnitude spectrum instead of the power spectrum can lead to different results. Other implementation differences, however, can be more substantial. Most commonly found are the use of a different frequency transform, the selection of spectral content to aggregate, and the normalization.

With respect to frequency transforms, the CQT as introduced in Section 3.3.2 can also be used for pitch chroma computation [114]. In this case, the number of bands per semitone will equal 1 or be constant. A modification of the STFT has been proposed by Cremer and Derboven [115] for the computation of the pitch chroma: they utilize a so-called “frequency-warped” STFT as introduced by Oppenheim et al. which uses a chain of first-order, all-pass filters to achieve nonequidistant spacing of the frequency bins [116]. Alternative frequency transforms such as resonance filterbanks (see Figure 3.15) have been proposed but did not find widespread adoption for pitch chroma extraction.

When using a STFT as frequency transform, different windows with triangular, trapezoid, or sinusoidal shapes can be applied to each semitone band to weight bins in the center of the band higher than bins at the boundaries, as opposed to the rectangular window implicitly used in Eq. (7.45) and plotted in Figure 7.29 [117]. Frequency components at the boundaries of each band might be noise-like components or high harmonics and thus not necessarily of interest for the pitch chroma extraction. To focus the extraction on tonal components, the pitch chroma can also be computed from a peak-picked or tonalness-weighted magnitude spectrum (compare, e.g. Section 3.5.11) to focus only on salient, tonal bins. Instead of using expert knowledge to parametrize the pitch chroma computation manually, chroma features can also be learned directly from data [118].

While the instantaneous pitch chroma per STFT block as a snapshot of the instantaneous pitch content can be used for analysis [119, 120], it is often considered to be too unstable to come to conclusions about the tonal content. Therefore, multiple pitch chromas are often aggregated. The window length for aggregation of the pitch chroma is another parameter allowing variations between different implementations. Some algorithms combine a fixed number of short analysis blocks [121], adapt the texture window length to the period between two neighboring beats [122], or adjust it to a measure of harmonic change in the audio [123]. The aggregation texture window length also needs to be adapted to the task at hand; for example key detection requires comparably long windows as opposed to chord recognition requiring maximum window lengths between a beat and a bar length.

Various pitch chroma implementations have been compared in the context of chord recognition by Jiang et al. [124].

7.5.1.1 Pitch Chroma Properties

The widespread use of the pitch chroma makes a detailed discussion and summary of its properties and characteristics worthwhile. In most implementations, the pitch chroma is assumed to only represent fundamental frequencies (and no overtones). In reality, all frequency content in the predefined frequency range of interest is mapped to the pitch chroma, regardless of it being a fundamental frequency or a higher harmonic. This leads to two possible problems:

- High nonpower-of-two harmonics lead to distortions in the pitch chroma by adding undesired components. Figure 7.31 displays a model series of 10 harmonics (amplitude weighting $1/k$) with the pitch $A3$ (top) and the resulting pitch chroma (bottom). The pitch chroma shows spurious components at E and $C\sharp$ and to a lesser degree at G and B. It is possible to partly compensate for this effect by using an amplitude model for the harmonics and modifying the pitch chroma accordingly [114]. The effect can also be attenuated by applying a weighting function before pitch chroma extraction to de-emphasize higher frequencies (see e.g. [125]).
- High harmonics may even be able to distort the pitch chroma in a different way: since the harmonics will deviate from the equal temperament – the temperament the pitch chroma computation is based on – they may map to pitches not really part of the tonal context. This effect is, however, of limited influence since the deviation of the lower harmonics is relatively small as shown in Table 7.6 and the magnitude of high harmonics is commonly low.

One possibility to avoid these artifacts is to use a multi-pitch detection system as a preprocessing step which ensures that only fundamental frequencies are mapped to the pitch chroma (compare [121, 126, 127]). The requirement of a reliable

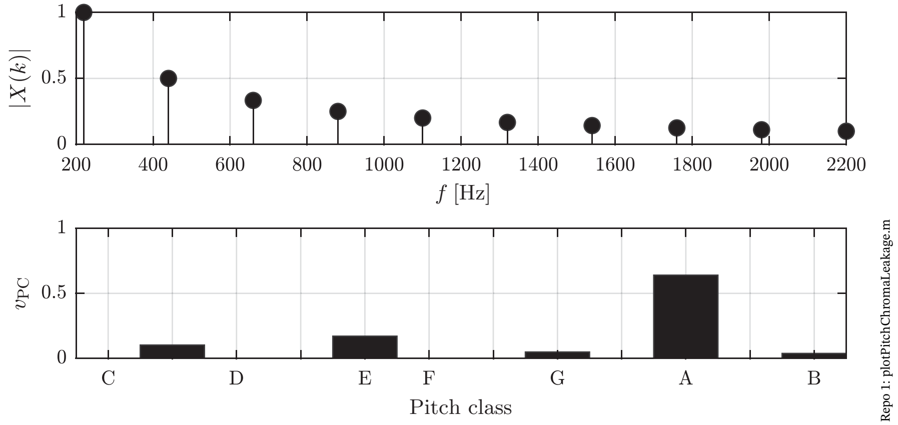


Figure 7.31 Pitch chroma of a hypothetical pitch $A3$ with 10 harmonics.

Table 7.6 Deviation (in Cent) of seven harmonics from the nearest equal-tempered semitone mid-frequency.

Harmonic	$\|\Delta C(f, f_T)\|$
$f = f_0$	0.00
$f = 2 \cdot f_0$	0.00
$f = 3 \cdot f_0$	1.96
$f = 4 \cdot f_0$	0.00
$f = 5 \cdot f_0$	13.69
$f = 6 \cdot f_0$	1.96
$f = 7 \cdot f_0$	31.17
$\mu_{\|\Delta C\|}$	6.97

multi-pitch detection system, however, might counterbalance the advantages of the pitch chroma as a comparably robust and simple-to-extract, low-level feature.

Several authors proposed to resort the pitch chroma entries to reflect (key) relationships between the pitch classes. One way to do so is to replace the standard modulo operation for computing the pitch chroma index as given in Eq. (7.13) by a shifted version:

$$\mathrm{PC_S}(\mathfrak{p}) = \mathrm{mod}\,(7 \cdot \mathfrak{p}, 12). \tag{7.49}$$

This results in related keys with a distance of seven semitones (i.e. a fifth) to be close to each other. Table 7.7 shows the pitch class order of the rearranged chroma. Certain similarity measures such as the Cross Correlation Function (CCF) or the cosine similarity will behave more gracefully or musically meaningfully when using this resorted pitch chroma [97, 113]. Note, however, that the rearranged pitch chroma does not take into account the cyclic nature as indicated by the circle of fifths unless specifically modeled (e.g. by using the Circular Correlation Function (CiCF)).

7.5.1.2 Features Derived from the Pitch Chroma

Similar to extracting features from a spectrum or a distribution, derived features can be extracted from the 12-dimensional pitch chroma as well. There is no

Table 7.7 Pitch class order in the original and the rearranged pitch chroma.

PC	C	$C\sharp$	D	$D\sharp$	E	F	$F\sharp$	G	$G\sharp$	A	$A\sharp$	B
$\mathrm{PC_S}$	C	G	D	A	E	B	$F\sharp$	$C\sharp$	$G\sharp$	$D\sharp$	$A\sharp$	F

established set of features to be extracted from the pitch chroma, although the set of three features presented by Tzanetakis et al. proved to be simple yet effective [113]. They use both the index and the amplitude of the pitch chroma maximum as well as interval between its two highest bins. For the latter feature, the pitch chroma is rearranged in order to place related keys as neighbors (see above).

There exists a multitude of other features to extract from a pitch chroma that might be useful in certain applications. Possibilities include a pitch chroma crest factor or the centroid of the resorted pitch chroma. The computation of statistical features such as the standard deviation is only of limited use due to the small number of elements.

7.5.2 Approaches to Key Detection

Key detection approaches are commonly based on the two assumptions that (i) the occurrence of key inherent pitch classes is more likely than the occurrence of non-key pitch classes and that (ii) the relative number and energy of occurrences is an indication of the key. When detecting the key of pieces with modulations, i.e. of pieces with a changing key, analyzing only small sections at the start and end of the piece of music will improve results as it is likely that the piece will start and end in the same (main) key [100, 125, 128].

Following the basic structure of an ACA system (see Figure 2.1), automatic key detection systems are usually based on two steps:

1. the extraction of some kind of pitch class distribution such as the average pitch chroma, and
2. the estimation of the most likely key given this distribution.

Most approaches attempt to identify two modes with 12 tonics each, leading to 12 major and 12 minor keys.

Similar to many other tasks, modern key detection approaches tend to be end-to-end systems based on DNNs [129, 130]; however, traditional approaches based on pattern recognition are still in use. A prototypical approach of detecting the musical key is introduced in the following. First, the pitch chroma as introduced in Section 7.5.1 is extracted. The timeframe over which the pitch chroma is averaged can vary from the whole file to a shorter texture window length to accommodate local key changes or to allow for multiple key estimates per file and doing a majority vote to yield the final result. It is also possible to extract the pitch chroma only at the beginning and end of the audio file because the key is usually less ambivalent in these regions [100]. The second step estimates the most likely key given a *template* pitch chroma, in the following referred to as *key profile*, by finding the minimum distance between the extracted pitch chroma and all key profiles.

7.5.2.1 Key Profiles

The *key profile* is a pitch class distribution used as reference for each key. The assumption is that the key profiles of identical modes nevertheless different tonics are identical but (circularly) shifted.

The key profile templates can be constructed in various ways, and Table 7.8 shows a few possible templates. The most trivial approach is the *Orthogonal* template v_o. The orthogonal template assumes that the tonic is the most salient component of the pitch chroma. The two main disadvantages are that this template allows no distinction between major mode and minor mode and that all other keys have the same distance (and are thus considered unrelated).

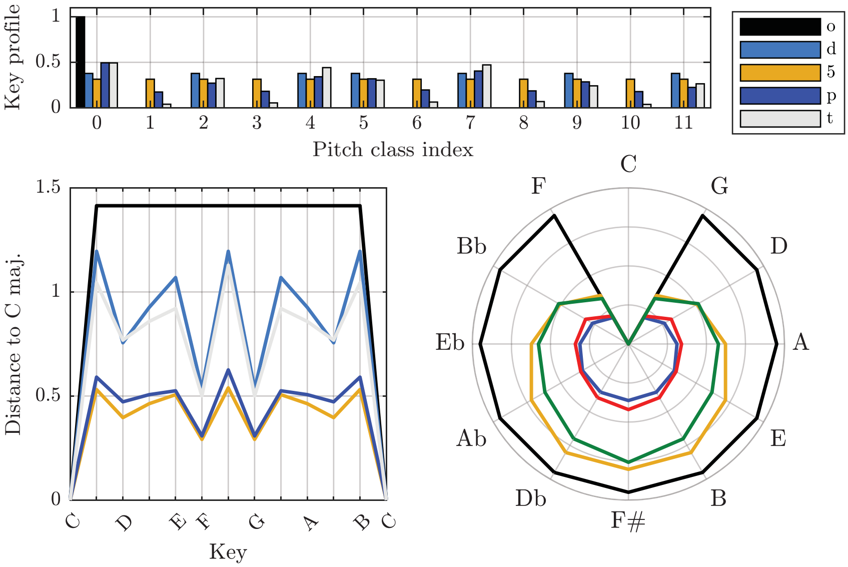
The latter problem can be solved by using *Diatonic* template v_d, which sets all key-inherent pitch classes to nonzero values. This template is musically sound as key distances increase linearly like an "unwrapped" circle of fifths; however, it still does not allow differentiation between major mode and minor mode.

The distance of two keys can also be modeled by a geometric model of the circle of fifths with radius r, represented by the template v_5. The coordinates of each key are given by the angle of the key in the circle of fifths. Each key has the same distance r to the point of origin. The model can be expanded into a third dimension to also include a distance between major mode and minor mode.

The key profile can also be based on tonal perception like the frequently used *Probe Tone Ratings* v_p. Although Krumhansl's probe tone ratings are not directly a key description but the result of a listening test evaluating how a specific scale degree fits into a given tonal context [110], Krumhansl showed in experiments that

Table 7.8 Various key profile templates, normalized to a vector length of 1.

	v_o	v_d	v_5	v_p	v_t
$v(0)$	1	0.38	$r \cdot e^{j2\pi\frac{0}{12}}$	0.49	0.49
$v(1)$	0	0	$r \cdot e^{j2\pi\frac{-5}{12}}$	0.17	0.04
$v(2)$	0	0.38	$r \cdot e^{j2\pi\frac{2}{12}}$	0.27	0.32
$v(3)$	0	0	$r \cdot e^{j2\pi\frac{-3}{12}}$	0.18	0.054
$v(4)$	0	0.38	$r \cdot e^{j2\pi\frac{4}{12}}$	0.34	0.44
$v(5)$	0	0.38	$r \cdot e^{j2\pi\frac{-1}{12}}$	0.32	0.3
$v(6)$	0	0	$r \cdot e^{j2\pi\frac{6}{12}}$	0.2	0.063
$v(7)$	0	0.38	$r \cdot e^{j2\pi\frac{1}{12}}$	0.4	0.47
$v(8)$	0	0	$r \cdot e^{j2\pi\frac{-4}{12}}$	0.19	0.069
$v(9)$	0	0.38	$r \cdot e^{j2\pi\frac{3}{12}}$	0.29	0.24
$v(10)$	0	0	$r \cdot e^{j2\pi\frac{-2}{12}}$	0.18	0.038
$v(11)$	0	0.38	$r \cdot e^{j2\pi\frac{5}{12}}$	0.22	0.26



Repo 1: plotKeyProfiles.m

Figure 7.32 Key profile vectors (top left) and the resulting interkey distances (major) to *C Major* (bottom left and right).

these probe tone ratings correlate well with the number of occurrences of the pitch classes in real pieces of music.

Finally, a data-driven approach leads to the *Extracted Key Profiles* v_t, here in an example proposed by Temperley extracted from symbolic data [131]. Note that average key profiles can also be extracted from audio and might be genre-specific [132].

The introduced key profiles are shown in both Table 7.8 and in Figure 7.32 (top).

7.5.2.2 Similarity Measure between Template and Extracted Vector

The most likely key can be estimated by finding the minimum distance between the extracted pitch chroma v_e and the key profile template. Under the reasonable assumption that the template key profiles for different keys are identical but shifted versions of the *C Major* profile, only one template needs to be stored per mode. Detecting modes other than major mode and minor mode is often considered not to be of relevance in practice; thus, only two template vectors have to be stored to generate an overall set of 24 shifted templates.

The index j of the most likely key can then be computed by finding the minimum distance d between the extracted pitch chroma v_e and the set of 24 template key profiles v_t:

$$j = \min_{0\leq s<24} \left(d(s)\right). \tag{7.50}$$

Typical distance measures can be the Euclidean distance or the Cosine distance (see Section 4.4). In the latter case, the identification of the key can also be expressed as a circular CCF.

The distances between different keys not only depend on the distance measure itself but also on the predefined key profile. Figure 7.32 plots the interkey distances between the shifted key profile templates and *C Major*; the distances are shown both in a line plot (bottom left) and arranged similar to the circle of fifths (bottom right). The Euclidean distance has been used to compute these diagrams; therefore, the input key profiles have been normalized to a vector length of 1 (as opposed to a sum of 1). The distances generally behave as anticipated with the distance to *C Major* being the minimum. While the orthogonal template has a high distance to other keys, it does not allow to differentiate between related and less related keys. All remaining profiles basically show the tendency of greater distances for keys more distant according to the circle of fifths. This is demonstrated by the cardioid shape of these distances in the polar diagram. It should be pointed out that the graph only visualizes the distances for keys in major mode and that none of the templates but the probe tone ratings and the extracted key profiles are able to separate between major mode and minor mode keys (although the model based on the circle of fifths could be adapted to a 3D model).

Theoretically, the use of these vector-based inter-key distance measures is questionable as the pitch chroma is a distribution and no vector. However, vector distances have been used and appear to work reasonably well in the absence of other fitting distance metrics.

It is possible to smooth the pitch chroma to avoid maximum distances for nearby pitches [133] which in the case of tonality perception is probably most effective if the pitch chroma is reordered in a way that related keys have similar pitch chroma indices as described in Section 7.5.1.1.

There are other more complex mathematical models for estimating a distance between two keys or pitch class distributions; one example is Chew's *spiral array model* [134, 135]. Another model targeted specifically at automatic tonality detection is the multidimensional *tonal centroid* proposed by Harte et al. [123] for which the pitch chroma is converted into a six-dimensional representation called tonal centroid based on the so-called "harmonic network" or *Tonnetz* which – under the assumption of enharmonic equivalence – can be transformed into three two-dimensional planes representing the circle of fifths, major thirds, and minor thirds. Another model is Gatzsche's *circular pitch space* [136].

7.5.3 Evaluation

As the musical key is a common annotation in Western music, ground truth data can be more easily accessible than for other tasks. Western classical music, for example, often uses the key as one identifier next to the name. The caveat in this case, however, is that the key might change frequently throughout the piece depending on form, genre, and epoch. These modulations break the assumption of one, static key per piece of music that many approaches are based on. For popular music, this assumption is much more likely to be correct, and there exist Internet databases for looking up the keys of various songs.

7.5.3.1 Metrics

As key detection is a classification task, normal classification-based metrics can be used (see Section 6.1.1). The most commonly used metric is the accuracy. Occasionally, a weighted accuracy score is used that aims to account for the severeness of errors by treating the erroneous classification into related keys as a partly correct result. The following weightings have been proposed: 1.0 if the key is correctly detected, 0.5 if the estimated key is one-fifth above the ground truth key, 0.3 if the estimated key is a relative key (sharing the same pitch classes) of the ground truth key, and 0.2 if the estimated key is a parallel key (sharing the same tonic) of the ground truth key. The reasoning or heuristics behind these weightings have been lost in history, but it is worth pointing out that they do not necessarily match the errors that typical key detection systems produce (compare, e.g. [100, 129]).

7.5.3.2 Datasets

Despite the availability of ground truth annotations, the number of complete audio key detection datasets remains limited. As the task of musical key detection requires the whole or a significant segment of the song, researchers cannot easily release commercial real-world data without violating licensing restrictions. For example, most of the key data available on Isophonics[10] does not come with associated audio data, only track information. Focusing on electronic dance music, the GiantSteps dataset [137] provides key labels. There are also datasets for other tasks that have been annotated with musical keys by other researchers, many of such are listed online on the accompanying website.[11]

7.5.3.3 Results

As with other tasks, the results vary largely between different datasets. State-of-the-art systems seem to be able to yield an accuracy in the general range of 70–90%.[12]

---

[10] Isophonics datasets. http://isophonics.net/datasets. Last retrieved on 6 August 2021.

[11] Audio Content Analysis Datasets. https://www.AudioContentAnalysis.org/data-sets. Last retrieved on 6 August 2021.

[12] MIREX 2019. https://www.music-ir.org/mirex/wiki/2019:Audio_Key_Detection_Results. Last retrieved on 6 August 2021.

7.6 Chord Recognition

The simultaneous use of (usually no less than three) different pitches creates a chord. Chords built of three pitches are referred to as triads. The most common chord types are built of third intervals – a few examples are displayed in Figure 7.33 with respect to a root note of *F*. Note that the same naming convention is used to label both the key and some common chords (e.g. *A Major*), so one has to derive from the context which of the two is meant.

Chord-inherent pitches can be doubled in different octaves without changing the chord type. Most frequently, the chord's root note is doubled. If the lowest note of a sounding chord is not its root note, this chord is called *inverted*. The first chord inversion of a *C Major* triad would therefore have the pitch classes (bottom-up) *E-G-C*. Figure 7.34 shows the two possible inversions of a *D Major* triad. The second inversion tends to appear less frequently than the first.

Each chord can have one or more musical (harmonic) functions depending on the musical key as well as dependent on other musical context. The chord sharing the root note with the current key is referred to as the *tonic chord* and most commonly represents the tonal center. Other important harmonic functions are the so-called *dominant chord* with the key's fifth scale degree as the root note and the *subdominant chord* with the key's fourth scale degree as the root note. Dominant chords often induce an expectancy of a following tonic chord in the listener.

7.6.1 Approaches to Chord Recognition

Detecting the sequence of chords from a musical audio signal can be challenging, because the signal will not only contain the pitched chord content, but also

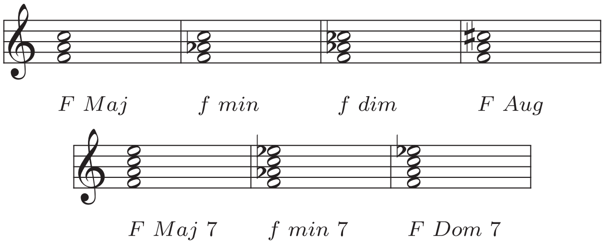


Figure 7.33 Common chords in musical score notation on a root note of *C*.

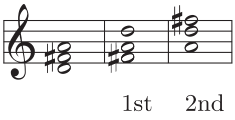


Figure 7.34 The two inversions of a *D Major* triad in musical score notation.

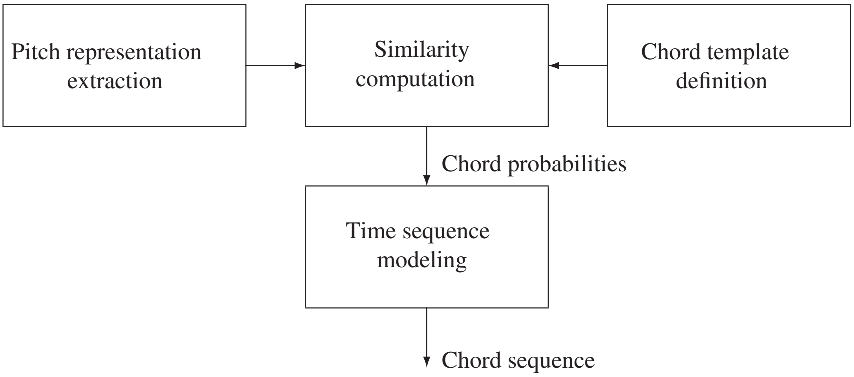


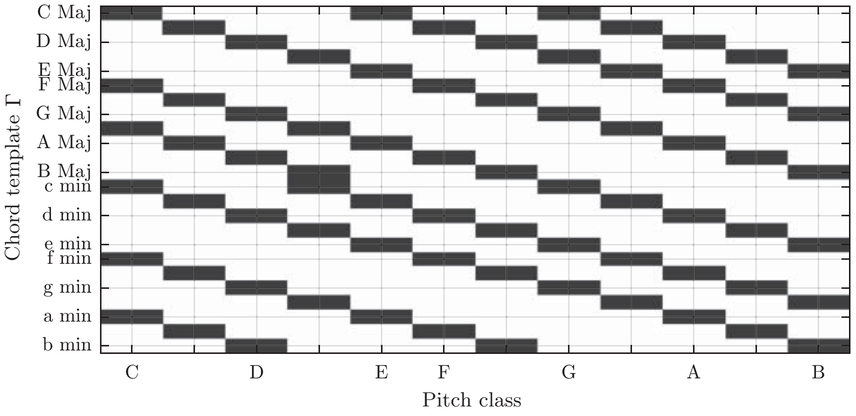
Figure 7.35 Flowchart of a simple chord detection system.

noisy content, e.g. from drums, and nonchord tones in other voices. Furthermore, a chord might be present in music without the pitches actually sounding simultaneously, for example, in the case of arpeggiated chords. This makes the recognition of chords a nontrivial problem.

The general structure of a typical *chord recognition* system is visualized in Figure 7.35. Pauwels et al. note in a recent survey article [138] that this general structure goes back to the method and discussion in Fujishima's seminal paper on chord recognition [111]. Cho and Bello investigate the impact of variations within the different stages on the overall chord recognition accuracy [139]. As with nearly as all ACA tasks, many newer approaches to this task cannot easily be structured in this way as they follow an end-to-end approach with DNNs [140–144].

Similar to key detection systems, the automatic recognition of chords most frequently utilizes the pitch chroma (see Section 7.5.1). Unlike in key detection systems, the pitch chroma is extracted from short segments in the piece of music as the focus is on the local tonal context as opposed to the global context. Some researchers propose to extract one pitch chroma for the period between each pair of neighboring beats [122, 145]. Although chords can obviously change anytime, changes tend to be more likely on the beat position, and the long aggregation window typically increases the robustness of the pitch representation.

The simplest chord template is a binary template in which only the chord-inherent pitches are nonzero; all templates can be grouped in a matrix $\mathbf{\Gamma}$ with the dimensions $\mathcal{T} \times 12$ with $\mathcal{T}$ being the number of chord templates and 12 pitch classes. Each row of the matrix represents one chord template; the trivial template presented here weights each pitch which is part of the chord by 1 and the remaining pitches with 0. Each row can then be normalized to the sum of all entries in this row. This leads to the template for the *C Major* triad to be



Repo 1: plotChordTemplates.m

Figure 7.36 Simple chord template matrix for chord estimation based on pitch chroma.

$[1/3, 0, 0, 0, 1/3, 0, 0, 1/3, 0, 0, 0]$. The complete chord template matrix is visualized in Figure 7.36. The number of chord templates is $\mathcal{T} = 24$ to detect major and minor triads. Other proposed methods for constructing chord templates are, for instance, based on a Tonnetz representation [123] or modeling the templates from data [146].

An estimate for the likelihood of each chord can then be estimated by computing a distance or similarity between the observed pitch chroma and the chord template. One way of doing so is with a linear transformation by the chord template matrix $\mathbf{\Gamma}$:

$$p_{\mathrm{E}}(n) = \mathbf{\Gamma} \cdot v_{\mathrm{pc}}(n). \tag{7.51}$$

This operation can also be interpreted as a computation of the correlation between pitch chroma and each template (each row of Γ). This is similar to the key detection approach mentioned in Section 7.5 if the cosine distance is used as distance measure. The resulting vector per block $p_{\mathrm{E}}(n)$ then has the dimension $\mathcal{T} \times 1$ and is a measure of the salience or likelihood of a specific chord given the pitch chroma $v_{\mathrm{pc}}(n)$.

These estimated likelihoods tend to be somewhat noisy and unreliable, especially in the case of short block sizes for pitch chroma extraction. There are various reasons for this, some of them technical (pitch chroma is only an imperfect representation of the tonal content), and some of them are musical (played or sung simultaneously sounding nonchord pitches, arpeggiations, and incomplete chords, etc.). This noisiness can be potentially reduced by smoothing the results with a low-pass (or median) filter applied to each chord likelihood [147]. Much more common, however, has been the use of the Viterbi algorithm which not only has the ability to smooth the results but also to model transitions between neighboring chords. Thus, modeling the chord progression with a hidden Markov model (HMM) has repeatedly been proven to increase stability of the results. The Viterbi algorithm as applied to chord recognition is introduced in detail below in Section 7.6.2. This postprocessing step can be replaced in neural networks by using, e.g. recurrent architectures [141].

Listing 7.9 shows the implementation of a simple blockwise chord estimation utilizing the introduced template. The implementation computes two results, one without postprocessing and one applying the Viterbi algorithm to the output.

```
% chord templates
[T] = generateTemplateMatrix_I ();

% transition probabilities
[P_T] = getChordTransProb_I ();

% pre-processing: normalization
x = ToolNormalizeAudio(x);

% extract pitch chroma
[v_pc, t] = ComputeFeature (’SpectralPitchChroma’,...
                            x,...
                            f_s,...
                            [],...
                            iBlockLength,...
                            iHopLength);

% estimate chord probabilities
P_E = T * v_pc;
P_E = P_E ./ sum(P_E, 1);

% assign series of labels/indices starting with 0
aiChordIdx = zeros(2, length(t));
[~, aiChordIdx(1,:)] = max(P_E, [], 1);

% compute path with Viterbi algorithm
[aiChordIdx(2,:), ~] = ToolViterbi(P_E,...
                            P_T,...
                            ones(length(cChords),1)/length(cChords),...
                            true);
```

Repo 2: computeChords.m
Repo 3: computeChords.py

Listing 7.9 Example implementation of a simple chord detection approach with a chord template matrix.

7.6.2 Viterbi Algorithm

The *Viterbi algorithm* provides a path-finding solution that identifies the globally most likely sequence of predefined states [148], given observations. The algorithm uses dynamic programming principles and is task-agnostic; here, we will focus on its use for chord recognition, but the principle can be easily transferred to other applications. The Viterbi algorithm is often considered an integral part of an HMM. It uses the following variables (the dimensionality given is based on the previous example – 12 pitch chroma dimensions with $\mathcal{N}$ observations and 24 chord templates):

- *States*: The sequence of states is being estimated and thus unknown (or *hidden*). In the case of chord recognition, each state will be a chord to be estimated (i.e. 24 states in the example above), and the sequence resembles the progression of chords.
- *Observations*: The extracted features are the observations. In our example, each pitch chroma will be one observation, resulting in an observation matrix $\boldsymbol{V}_{\mathrm{pc}}$ with dimensions $12 \times \mathcal{N}$.
- *Emission probabilities*: The emission probabilities model the relationship between observations and states, more specifically, the probability of an observation given a specific state. It is the likelihood of a measured pitch chroma, given a specific chord. The emission probabilities can be captured in an emission probability matrix $\boldsymbol{P}_{\mathrm{E}}$ with dimensions $24 \times \mathcal{N}$.
- *Start probabilities*: Different states might be more likely to occur than others as initial state. For example, in a key dependent chord recognition model of popular music, the tonic chord is more likely than other chords to appear at the song start. The start probabilities will be a vector p_{S} with dimensionality 24.
- *Transition probabilities*: The fact that transitions between certain states are more common than between others is modeled with the transition probabilities. For example, the transition from *A Major* to *D Major* can be expected to have a higher probability than the transition from *A Major* to *D♯ Major*. The transition probabilities for each pair of chords can be captured in a transition probability matrix $\boldsymbol{P}_{\mathrm{T}}$ with dimensions 24×24.

For an implementation, we furthermore need an empty accumulated probability matrix $\boldsymbol{P}_{\mathrm{res}}$ storing the highest likelihood to reach a state, and an empty backtracking matrix $\boldsymbol{I}$ storing the most likely preceding state for each state with dimensions $24 \times \mathcal{N}$.

The model for the (emission, transition, start) probabilities at the core of the HMM is usually derived from training data, although that is not necessarily the case. The training data can either be annotated audio [122], MIDI-synthesized audio [149], or symbolic data [119]. Without data, musicologically driven models can be used: A very simple (but key independent) model for the chord progression likelihood is to use a circle of fifths [122]. A related approach is to utilize the correlation between Krumhansl's key profiles as chord distances [119]. Note that in these cases, we model the chord relations through key relations, which is systematically questionable. The preference for specific keys and root notes in specific genres often makes training data augmentation through pitch shifting (Section 5.2) desirable.

The Viterbi algorithm identifies the most likely state sequence given a set of observations by estimating the probability of each state at each time instance depending on all previous state probabilities. This avoids the calculation of every possible path through the state sequence. The following steps are executed (with reference to the presented simple chord detection):

1. Initialize transition and start probabilities through model (usually data driven). Here: algorithmic through the circle of fifths.
2. Compute observations. Here: the pitch chroma.
3. Compute emission probabilities (usually through a GMM). Here: modeled through applying Eq. (7.51) $\boldsymbol{P}_\mathrm{E} = \boldsymbol{\Gamma} \cdot \boldsymbol{V}_\mathrm{pc}$.

The remaining steps are task-agnostic:

4. Initialize the first column of state probabilities given the start probabilities:

   $$P_\mathrm{res}(s, 0) = p_\mathrm{S}(s) \odot P_\mathrm{E}(s, 0) \tag{7.52}$$

   with $\odot$ being elementwise multiplication (Hadamard product).
5. Iterate from the second column over all columns n:

   $$P_\mathrm{res}(s, n) = P_\mathrm{E}(s, n) \cdot \max_{\forall j} \left(P_\mathrm{E}(j, n-1) \cdot P_\mathrm{T}(j, s)\right) \tag{7.53}$$

   $$I(s, n) = \operatorname*{argmax}_{\forall j} \left(P_\mathrm{E}(j, n-1) \cdot P_\mathrm{T}(j, s)\right). \tag{7.54}$$

   The probability to reach each state s at time n depends on the state's emission probabilities and the most likely previous state as modeled by the previous state's probability and its transition probability to state s. The index of this preceding state is remembered in the index matrix $\boldsymbol{I}$.
6. The best final state is then

   $$s_\mathrm{final} = \operatorname*{argmax}_{\forall s} \left(P_\mathrm{res}(s, \mathcal{N} - 1)\right). \tag{7.55}$$

7. Finally, the path through the emission probability matrix with the highest probability can be backtracked from by walking back to the left using the indices stored in $\boldsymbol{I}$ starting from $I(s_\mathrm{final}, \mathcal{N} - 1)$.

Listing 7.10 shows an implementation of this algorithm.

For long sequences, the probabilities $\boldsymbol{P}_\mathrm{res}$ will approach zero. The most common solution is the use of the logarithmic likelihood. Alternatively, each column could also be normalized to a sum of 1 during the iteration if the overall sequence probability is not of interest.

Figure 7.37 visualizes the impact of the Viterbi algorithm with a chord detection example. In this example, the emission probabilities are computed through the

```
        % initialization
        P_E = log(P_E); % hope for non-zero entries
        P_T = log(P_T); % hope for non-zero entries
        p_s = log(p_s); % hope for non-zero entries
        I = zeros(size(P_E));
        P_res = zeros(size(P_E));

        P_res(:, 1) = P_E(:, 1) + p_s;

        % recursion
        for n = 2:size(P_E, 2)
            for s = 1:size(P_E, 1)
                % find max of preceding times trans prob
                [p_max, I(s, n)] = max(P_res(:, n-1) + P_T(:, s));
                P_res(s, n) = P_E(s, n) + p_max;
            end
        end
    end

    % traceback
    p = zeros(1,size(P_E, 2));
    % start with the last element, then count down
    [prob, p(end)] = max(P_res(:, end));
    for n = size(P_E, 2)-1:-1:1
        p(n) = I(p(n+1), n+1);
    end
```

Repo 2: ToolViterbi.m
Repo 3: ToolViterbi.py

Listing 7.10 Example implementation of the Viterbi Algorithm with logarithmic likelihoods.

template matrix as described above, and the transmission probabilities are approximated through key distances with the circle of fifths. The figure shows the first 20 seconds of a pop song; the emission probabilities are shown on the top left, and the resulting chord sequence *without* Viterbi postprocessing is shown in the bottom left; the top right shows the used chord transition matrix, and the bottom right displays the output of the Viterbi algorithm. Comparing the two paths, we see that applying Viterbi smooths the path considerably, especially in the case of high self-transition probabilities. The resulting chords on the bottom right equal the ground truth for the first 10 seconds. The detection gets somewhat erroneous when the vocals start after that.

Note that the Viterbi algorithm follows the same dynamic programming principles as Dynamic Time Warping (DTW) (see Section 10.1); in some ways, it can be seen as a generalization of DTW. In both cases, the goal is to find the path through a matrix which maximizes the probability or minimizes the cost. The main differences between DTW and the Viterbi algorithm are (i) that DTW has additional constraints on the path, which has to start in one corner of the matrix, end in the opposite corner, and move only to neighboring elements, constraints that the Viterbi algorithm models more generally through start and transition

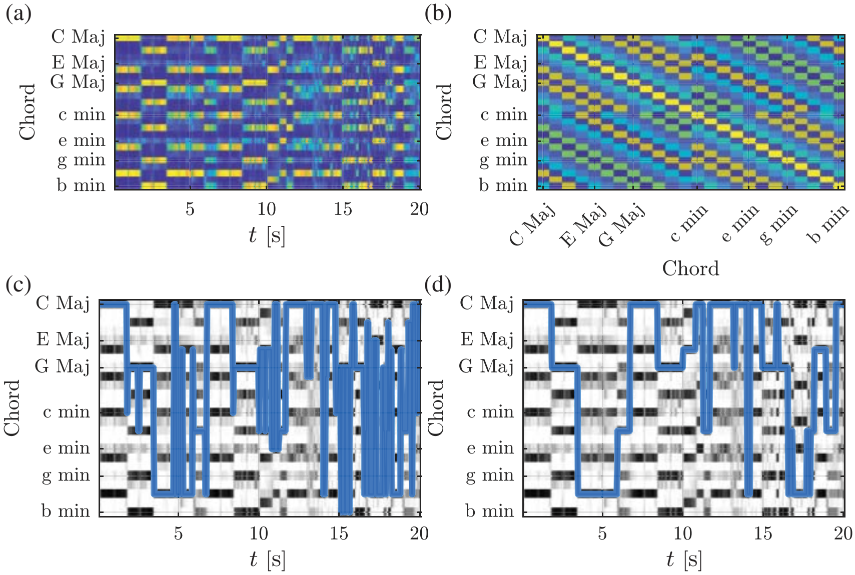


Figure 7.37 Chord emission probability matrix (a), transition probability matrix (b), instantaneous chord detection results by taking the maximum emission (c), and output of the Viterbi algorithm (d).

probabilities, (ii) that DTW's cost matrix can be seen as the inverse of the accumulated probability matrix, (iii) that neither the transition cost of DTW nor the construction of its distance matrix are commonly driven by training data unlike the emission and transition probabilities, and (iv) that the DTW path length is not predefined.

7.6.3 Evaluation

The objective for the evaluation of chord recognition systems is the comparison of the estimated labels with the ground truth annotations. This task is complicated by several challenges. First, the annotation of chord labels can be errorprone if human subjects identify the chords through listening. Second, chord labels can be subjective to a certain degree; for example there can be disagreement between annotators whether a melody pitch is considered part of a chord or not. Third, chords and chord types can be musically ambiguous without context, especially in the enharmonic settings commonly applied in ACA; depending on musical context, the same pitch chroma representation might represent different chords with different root notes. This can lead to considerable disagreements between annotators [150]. Fourth, not all types/segments of music contain chords (e.g. monophonic and percussive parts, counterpoint), complicating the labeling and recognition.

Given these challenges and subjectivities, the annotations in general cannot be considered absolute truth, complicating the systematic evaluation of chord detection systems [151].

Other difficulties arise from the definition of the chord vocabulary used and potential mismatches between the ground truth chord vocabulary and the chords detected by the system. If a system detects only the 24 major and minor triads as suggested above, how should it deal with ground truth annotations of a seventh chord? Can the seventh chord be reduced to a triad or should this annotation be removed from the evaluation? An additional problem is that the more detailed and complicated the chord annotations become, the more pronounced the imbalance between the various chords in the data will become. Burgoyne et al. report that, in a dataset of Billboard songs, just the seven most popular chords (*C Major*, *D Major*, *G Major*, *A Major*, *E Major*, *F Major*, *B♭ Major*) cover more than 30% of the observations and that the five most popular chord types cover more than 80% of the data [152].

7.6.3.1 Metrics

Similar to other tasks such as pitch detection, chords will be annotated with start and end times as opposed to a chord label per short time segment, the output format of a conventional chord detection system. Frequently, the ground truth is processed and quantized into equivalent time segments to allow for computation of classification metrics (see Section 6.1.1). The class imbalance requires the use of macromeasures to account for less-prominent chord classes.

The classification metrics do not necessarily have to be computed on the chord classes; it has been suggested to evaluate broader categories of the ground truth such as root note, chord type, chord inversion individually [153]. A different aspect of evaluation can be the time segmentation, more specifically a measure of the system tendency to over- or undersegment the chord labels.

7.6.3.2 Datasets

Although there is a reasonable amount of time-aligned annotated data available nowadays, most often the annotations come without audio data [150, 152, 154, 155] except for a minority of datasets [156, 157].

7.6.3.3 Results

The issues mentioned above make automatic chord recognition a challenging task with a room for improvement. Furthermore, the chord imbalance combined with the number of metrics and huge variations related to the test datasets make reporting a specific number range questionable. The interested reader is referred to previous MIREX campaigns for detailed results.[13]

---

[13] MIREX 2018. https://www.music-ir.org/mirex/wiki/2018:Audio_Chord_Estimation_Results. Last retrieved on 6 August 2021.

7.7 Exercises

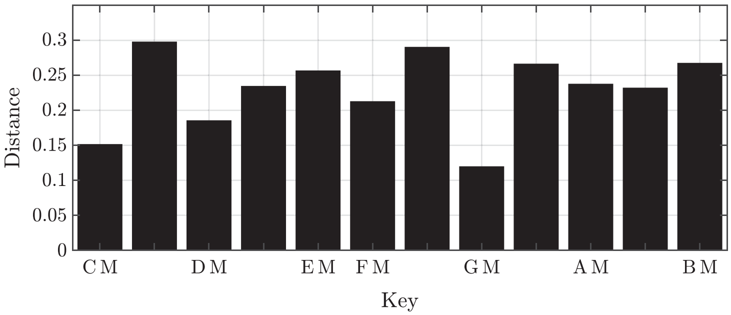
7.7.1 Questions

1. Describe the relationship between frequency and perceived pitch.
2. How many pitch classes is an octave divided into (for Western music)?
3. If the frequency of 440 Hz is mapped to the MIDI pitch 69, what is the MIDI pitch corresponding to a frequency of 880 Hz?
4. What is the pitch distance in Cent between the two frequencies 600 and 900 Hz?
5. Name three approaches traditionally used for monophonic fundamental frequency detection.
6. Why can preprocessing with a low-pass filter potentially increase the robustness of ACF-based fundamental frequency detection? How do you choose an appropriate cut-off frequency?
7. Consider the simplest implementation of an ACF-based fundamental frequency detection. If the first nonzero maximum occurs at lag 100, what is the estimated fundamental frequency? Assume a sample rate of 40 kHz.
8. You have a monophonic audio signal likely missing some harmonics. Would the HPS or the HSS be a better method to detect the fundamental frequency? Why?
9. The frequency resolution of FFT-based fundamental frequency detection determines the maximum frequency accuracy of your result. Is this potential problem more pronounced for lower-input frequencies or higher-input frequencies. Describe two ways of addressing this problem.
10. Which of the following signals might result in unreliable instantaneous frequency estimates?
    - an amplitude-modulated sinusoidal
    - two sinusoidals with very close frequencies
    - a high-frequency rectangular wave
11. Assuming a sample rate of 40 kHz and a hop size of 400 samples, what is the instantaneous frequency in Hz if the phase difference at one bin is $2\pi \cdot 0.01$?
12. List and highlight the main challenges of polyphonic pitch detection as compared to monophonic pitch detection.
13. Describe the general process of the “iterative subtraction” paradigm for polyphonic pitch detection.
14. Given a spectrogram of size 1025×2000 (frequency × blocks), name the dimensions of both the template matrix $\boldsymbol{W}$ and the activation matrix $\boldsymbol{H}$ when estimating $\hat{\boldsymbol{X}} = \boldsymbol{W}\boldsymbol{H}$ with NMF with a rank $r = 15$.

15. Given the following matrix, name one optimal (no error) factorization result $(\boldsymbol{W}, \boldsymbol{H})$.

$$\boldsymbol{X} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 10 & 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 15 & 15 & 0 \end{bmatrix}$$

16. Describe how a tuning frequency estimate might improve a pitch-based audio analysis task like chord detection.
17. What are the dimensions of the average pitch chroma computed over N blocks?
18. Given the distance plot below between an extracted pitch chroma and a shifted template, which is the most likely key of this signal?



19. You observe a pitch chroma vector that reads `v_pc = [0.0 0 0.35 0 0 0 0.26 0 0 0.29 0 0.1]`. What chord is most likely?
20. Is the following statement true? "If an emission probability is high for a given pair of observation and state, it ensures the occurrence of this state in the final decoded state sequence."
21. Consider a HMM with the two chords C, D as the hidden states and two pitch chroma observations $v_{\text{pc},1}, v_{\text{pc},2}$. Assume the initial state is *Start*, and we know the following parameters:

$$\begin{aligned} P(C \rightarrow D) &= 0.3 \\ P(D \rightarrow D) &= 0.2 \\ P(v_{\text{pc},1}|C) &= 0.1 \\ P(Cv_{\text{pc},2}|D) &= 0.4 \\ P(Start \rightarrow C) &= 0.9. \end{aligned}$$

- (a) What is $P(Start \rightarrow D)$?
- (b) What is $P(C \rightarrow C)$?
- (c) What is $P(D \rightarrow C)$?
- (d) What is $P(v_{\mathrm{pc},1}|D)$?
- (e) What is $P(v_{\mathrm{pc},2}|C)$?

7.7.2 Assignments

1. Blockwise pitch tracking with the ACF
   - (a) Blocking and fundamental frequency extraction
     - Implement a function `[r_xx] = comp_acf(x_block, bIsNormalized)` which computes the ACF for a single block `x_block` (of length `blockSize`) and returns the nonredundant (right) part of the result. Normalization is switched on and off by parameter `bIsNormalized`.
     - Implement a function `[f0] = get_f0_from_acf (r_xx, fs)` that takes the output of `comp_acf` and computes and returns the fundamental frequency estimate f_0 of that block in Hz.
     - Implement a function `[f0_B, t_B] = track_pitch_acf(x, blockSize, hopSize, fs)` returns two vectors `f0_B` and `t_B` containing the detected fundamental frequency per block and the time stamp of each block, respectively. The function utilizes the functions `read_audio`, `downmix_audio`, and `block_audio` implemented in Section 3.8.2 to read an audio file, downmix it if necessary, and segment it into overlapping blocks, as well as the functions `comp_acf` and `get_f0_from_acf`. If you haven't implemented the function from a previous assignment yet, follow the following instructions:
       - Implement a function `[fs, x] = read_audio(cPath2Audio)` which reads a wave audio file with scipy.io.wavfile.read and returns the sample rate `fs` and a numpy array `x` with dimensions `numSamples` × `numChannels`. Ensure that the maximum possible value range of `x` is always $[-1, 1[$ regardless of the wordlength of the input file. Support the following wordlengths/input types: `uint8`, `int16`, `int32`, `float32`
       - Implement a function `[x_m] = downmix_audio(x)` which takes a numpy array `x` with dimensions `numSamples` × `numChannels` and returns the downmixed numpy array `x_m` with dimensions `numSamples` × 1.
       - Implement a function `[x_B, t_B] = block_audio(x_m, blockSize, hopSize, fs)` which returns a numpy array `x_B` (dimension `N` × `blockSize`) and a vector t_B (dimension `N` × 1) for

blocking the input audio signal into overlapping blocks. Make sure that each sample of the audio signal is included in at least one of the `N` blocks. `t_B` will refer to the start time of each block in seconds.

(b) Testing and Evaluation
- Implement a function `test_acf_code()` for
  - generating a sinusoidal test signal of length 4 s at a sample rate $f_S = 44.1$ kHz with
    - 0–1 s: $f = 441$ Hz,
    - 1–2 s: $f = 882$ Hz,
    - 2–3 s: $f = 438.81$ Hz,
    - 3–4 s: $f = 873.27$ Hz,
  - calling your `track_pitch_acf` function with this test signal,
  - plotting both, the resulting f0 and the detection error in Hz of the output, and

  discuss possible reasons for the deviation.
- Download the Tonas dataset.[14]
- Implement a function `[pitchInMIDI] = convert_freq2midi (freqInHz, fA4=440)` that returns a variable `pitchInMIDI` of the same dimension as input `freqInHz`. Note that the dimension of `freqInHz` can be a scalar, a vector, or a matrix. `fA4` can be specified but is set to 440 Hz by default.
- Implement a function `[errCentRms] = eval_pitchtrack (estimateInHz, groundtruthInHz)` that computes the RMS of the error[15] in Cent (in the pitch domain, not frequency) between two vectors `estimateInHz` and `groundtruthInHz`. Ignore blocks with `groundtruthInHz == 0`.
- Implement a function `[errCentRms] = run_evaluation (path2Tonas)` returning the overall average error in cents as one value. This requires implementing a directory parser, file reader, and a loop calling your pitch tracker. Compare also the function `get_feature_data` from a previous assignment in Section 3.8.2. Note: the annotation form for Tonas is `[onset_seconds, duration_seconds, pitch_frequency, quantized_frequency]`.
- Run your implemented pitch tracker on the Tonas dataset and report the overall average `errCentRms`. Discuss whether the value is high or low and speculate why.

---

[14] TONAS. https://www.upf.edu/web/mtg/tonas. Last retrieved on 27 November 2021.

[15] The RMS of the error could also be referred to as the standard deviation with respect to the ground truth.

(c) Bonus
   - Implement a function `[f0_B, t_B] = track_pitch_acfmod(x, blockSize, hopSize, fs)` that modifies your method and improves the performance of your basic ACF pitch tracker thereby providing better f_0 estimations. You may apply constraints to your approach, preprocessing or postprocessing steps, or any other additional processing steps that might improve the result. Verify the improved result by running your evaluation.

2. Blockwise pitch tracking in the spectral domain
   (a) Fundamental frequency detection with the maximum of the magnitude spectrum
   - Implement a naive fundamental frequency detection in the frequency domain: `[f0_B, t_B] = track_pitch_fftmax(x, blockSize, hopSize, fs)` that estimates the fundamental frequency `f0` of the each audio signal block based on a blockwise maximum spectral peak finding approach. Note: This function can use `compute_spectrogram` from Section 3.8.2. If you haven't implemented it yet, use the following instructions:
     - Implement a function `[w] = compute_hann(windowLength)` that computes a Hann window of length `windowLength` and returns the window in numpy vector `w`. Before you implement, investigate the difference between choosing a "symmetric" vs. a "periodic" window function and choose the "periodic" implementation.
     - Implement a function `[X, fInHz, t_B] = compute_spectrogram(x_B, t_B, fs)` that computes the magnitude spectrum for each block of audio in `x_B` and returns the magnitude spectrogram `X` (dimensions `blockSize/2+1` × `N`), a frequency vector `fInHz` (dimensions `blockSize/2+1` × 1) containing the central frequency of each bin, and the previously computed time stamp vector `t_B`.

       For the computation of the spectrogram, use the Hann window computed with `compute_hann`. Scale your Fourier transform with `2/blockSize` to ensure the magnitude ranges don't vary between different window lengths. Note: remove the redundant part of the spectrum if necessary.
   - If the blockSize = 1024 in the above function, what is the resolution of your pitch tracker? Can this be improved without changing the block-size? If yes, how? If no, why? (Use a sampling rate of 44 100 Hz for all calculations).

   (b) Fundamental frequency detection with the HPS
      - Implement a function `[f0] = get_f0_from_Hps(X, fs, order)` that computes the blockwise fundamental frequency `f0` given the magnitude spectrogram X and the sampling rate based on a HPS approach of order `order`.
      - Implement a function `[f0_B, t_B] = track_pitch_hps(x, blockSize, hopSize, fs)` that estimates the fundamental frequency `f0_B` of each audio signal block based on the HPS-based approach. Use `compute_spectrogram` and an *order* = 4 for `get_f0_from_Hps`.

   (c) Evaluation
      - Evaluate your results with the function and data above. Try two parametrizations, a blockSize and hopSize of 1024 and 512, respectively, and a blockSize and hopSize of 8192 and 512, respectively.

3. Voicing detection

   (a) Voicing detection with an RMS threshold
      - Implement a function `[rmsDb_B] = extract_rms(x_B)` which takes the blocked audio as input and computes the RMS amplitude of each block (or reuse the function implemented in Section 3.8.2).
      - Implement a function `[mask_B] = create_voicing_mask (rmsDb_B, thresholdInDb)` which takes a vector of decibel values for the different blocks of audio and creates a binary mask based on the threshold parameter. Note: A binary mask in this case is a simple column vector of the same size as `rmsDb` containing 0's and 1's only. The value of the mask at an index is 0 if the `rmsDb_B` value at that index is less than `thresholdInDb` and the value is 1 if `rmsDb_B` value at that index is greater than or equal to the threshold.

   (b) Voicing evaluation
      - Implement a function `[f0Adj_B] = apply_voicing_mask(f0_B, mask_B)` which applies the voicing mask to the previously computed `f0_B` so that the f0 of blocks with low energy is set to 0.
      - Implement a function `[pfp] = eval_voiced_fp(f0EstInHz, f0GroundtruthInHz)` that computes the percentage of false positives for your fundamental frequency estimation. The denominator is the number of blocks for which annotation = 0. The numerator would be how many of these blocks were classified as voiced (with a fundamental frequency not equal to 0) is your estimation.
      - Implement a function `[pfn] = eval_voiced_fn(f0EstInHz, f0GroundtruthInHz)` that computes the percentage of false

negatives for your fundamental frequency estimation. In this case, the denominator is number of blocks which have nonzero fundamental frequency in the annotation, the numerator is number of blocks out of these that were detected as zero in the estimation.

- Modify the `eval_pitchtrack` method above to `[errCentRms, pfp, pfn] = eval_pitchtrack_v2(f0EstInHz, f0GroundtruthInHz)` to return all the 3 performance metrics for your fundamental frequency estimation. Note: the `errorCentRms` computation will have to be updated now to take into account that the estimated f0s can also contain zeros to be ignored.

4. Tuning frequency detection
   (a) Write a function `[spectralPeaks] = get_spectral_peaks(X)` that returns the top 20 spectral peak (local maxima) bins of each column of magnitude spectrogram `X`.
   (b) Write a function `[tfInHz] = estimate_tuning_freq(x, blockSize, hopSize, fs)` to return the tuning frequency. `x` is the time domain audio signal, `blockSize` is the block size, `hopSize` is the hopSize, and `fs` is the sample rate. Utilize the previously implemented functions `block_audio` and `compute_spectrogram` from Section 3.8.2, then call `get_spectral_peaks` to obtain the top 20 spectral peaks for each block. Then, compute the deviation in Cent of the peak bin from the closest equal-tempered pitch for each spectral peak. Then, compute a histogram of the deviations in Cent and estimate the tuning frequency from the location of the max of this histogram. Make sure your function works for any blockSizes.
   (c) Modify your function `estimate_tuning_freq` so that instead of using the frequency bin of each peak, use the instantaneous frequency at each peak bin (see Appendix B.6).
5. Key detection
   (a) Write a function `[v_pc] = extract_pitch_chroma(X, fs, tfInHz = 440)` which returns the pitch chroma array `v_pc` (dimensions 12 × numBlocks). `X` is the magnitude spectrogram, `fs` is the sample rate, and `tfInHz` is the tuning frequency. Compute the pitch chroma for the 3 octave range C3–B5. Make sure your implementation works with different sample rates and block sizes. You will need to adjust the semitone bands based on the input parameter `tfInHz`. Normalize each individual chroma vector to a length of 1 (L2 norm).
   (b) Write a function `[keyEstimate] = detect_key(x, blockSize, hopSize, fs, bTune)` to detect the key of a given audio signal `x`. The parameter `bTune` is `True` or `False` and will specify if tuning frequency

correction is applied before computing the key. Use the following template profiles for estimating the key (Krumhansl templates):

```
t_pc = np.array([    [6.35, 2.23, 3.48, 2.33, 4.38,
4.09, 2.52, 5.19, 2.39, 3.66, 2.29, 2.88],   #major
                     [6.33, 2.68, 3.52, 5.38, 2.60,
3.53, 2.54, 4.75, 3.98, 2.69, 3.34, 3.17]])  #minor
```

Note that this array contains both major and minor key profiles, respectively. Hint: don't forget to normalize the key profiles. Use Euclidean distance.

(c) Write a function `[accuracy] = eval_key_detection(path2Audio, path2GT)` that evaluates key detection for the audio files in `path2Audio`. For each file in the audio subdirectories, there will be a corresponding .txt file in the GT subdirectories containing the ground truth key label. Download the GTZAN genre dataset[16] and the ground truth key annotations[17] for your evaluation. You return the *accuracy* (number of correct key detections divided by total number of songs) of your key detection for all the files. Evaluate your algorithm with and without tuning frequency correction enabled.

References

1 Alain de Cheveigné. Pitch Perception. In Christopher J Plack, editor, *Oxford Handbook of Auditory Science: Hearing*, *Oxford Handbook of Auditory Science*. Oxford University Press, New York, January 2010. ISBN 978-0-19-923355-7. URL https://www.oxfordhandbooks.com/view/10.1093/oxfordhb/9780199233557.001.0001/oxfordhb-9780199233557-e-04.

2 Harvey Fletcher. The Physical Criterion for Determining the Pitch of a Musical Tone. *Physical Review*, 23(3):427–437, March 1924. ISSN 0031-899X. doi: 10.1103/PhysRev.23.427. URL https://link.aps.org/doi/10.1103/PhysRev.23.427.

3 Jan Frederik Schouten. The Perception of Subjective Tones. page 5, 1938.

4 Ben J Shannon and Kuldip K Paliwal. A Comparative Study of Filter Bank Spacing for Speech Recognition. In *Proceedings of Microelectronic Engineering Research Conference*, pages 2–4, Bisbane, November 2003.

5 Juan José Burred and Thomas Sikora. On the Use of Auditory Representations for Sparsity-Based Sound Source Separation. In *5th International Conference on Information Communications & Signal Processing*, pages 1466–1470, Bangkok, 2005. IEEE. ISBN 0-7803-9283-3.

---

[16] GTZAN. http://marsyas.info/downloads/datasets.html. Last retrieved on 27 November 2021.

[17] GTZAN-Key. https://github.com/alexanderlerch/gtzan_key. Last retrieved on Nov. 27, 2021.

doi: 10.1109/ICICS.2005.1689302. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1689302.

6 Stanley Smith Stevens, John Volkmann, and E B Newman. A Scale for the Measurement of the Psychological Magnitude Pitch. *Journal of the Acoustical Society of America (JASA)*, 8(3):185–190, 1937. ISSN 00014966. doi: 10.1121/1.1915893. URL http://link.aip.org/link/?JAS/8/185/1&Agg=doi.

7 Stanley Smith Stevens and John Volkmann. The Relation of Pitch to Frequency: A Revised Scale. *The American Journal of Psychology*, 53(3): 329, July 1940. ISSN 00029556. doi: 10.2307/1417526. URL http://www.jstor.org/stable/1417526?origin=crossref.

8 Robert J Siegel. A Replication of the Mel Scale of Pitch. *The American Journal of Psychology*, 78(4): 615, December 1965. ISSN 00029556. doi: 10.2307/1420924. URL http://www.jstor.org/stable/1420924?origin=crossref. 00019.

9 Gunnar Fant. *Speech Sounds and Features*. MIT Press, Cambridge, 1973. ISBN 0-262-06051-5.

10 Douglas O'Shaughnessy. *Speech Communication: Human and Machine*. Addison-Wesley, reprint edition, 1987. ISBN 978-0-201-16520-3.

11 John Makhoul and Lynn Cosell. LPCW: An LPC vocoder with linear predictive spectral warping. In *ICASSP '76. IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 466–469, April 1976. doi: 10.1109/ICASSP.1976.1170013.

12 S Umesh, L Cohen, and D Nelson. Fitting the Mel scale. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing. (ICASSP99)*, pages 217–220 vol. 1, Phoenix, 1999. IEEE. ISBN 0-7803-5041-3. doi: 10.1109/ICASSP.1999.758101. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=758101.

13 Eberhard Zwicker and Hugo Fastl. *Psychoacoustics. Facts and Models*. Springer, 2nd edition, 1999. ISBN 978-3-540-23159-2.

14 M R Schroeder, B S Atal, and J L Hall. Optimizing Digital Speech Coders by Exploiting Masking Properties of the Human Ear. *Journal of the Acoustical Society of America (JASA)*, 66(6): 1647–1652, 1979.

15 Ernst Terhardt. Calculating Virtual Pitch. *Hearing Research*, 1:155–182, 1979.

16 Eberhard Zwicker. Analytical Expressions for Critical-Band Rate and Critical Bandwidth as a Function of Frequency. *Journal of the Acoustical Society of America (JASA)*, 68(5): 1523–1525, 1980. ISSN 00014966. doi: 10.1121/1.385079. URL http://link.aip.org/link/JASMAN/v68/i5/p1523/s1&Agg=doi.

17 Hartmut Traunmüller. Analytical Expressions for the Tonotopic Sensory Scale. *Journal of the Acoustical Society of America (JASA)*, 88(1): 97–100, 1990. ISSN 00014966. doi: 10.1121/1.399849. URL http://link.aip.org/link/?JASMAN/88/97/1.

18 Brian C J Moore and Brian R Glasberg. Suggested formulae for calculating auditory-filter bandwidths and excitation patterns. *Journal of the Acoustical Society of America (JASA)*, 74(3): 750–753, 1983.

19 Ernst Terhardt. The SPINC Function for Scaling of Frequency in Auditory Models. *Acustica*, 77:40–42, 1992.

20 Donald D Greenwood. Critical Bandwidth and the Frequency Coordinates of the Basilar Membrane. *Journal of the Acoustical Society of America (JASA)*, 33(10):1344–1356, 1961. ISSN 00014966. doi: 10.1121/1.1908437. URL http://link.aip.org/link/JASMAN/v33/i10/p1344/s1&Agg=doi.

21 Roger N Shepard. Circularity in Judgments of Relative Pitch. *Journal of the Acoustical Society of America (JASA)*, 36(12):2346, 1964. ISSN 00014966. doi: 10.1121/1.1919362. URL http://link.aip.org/link/JASMAN/v36/i12/p2346/s1&Agg=doi.

22 Diana Deutsch, Kevin Dooley, and Trevor Henthorn. Pitch Circularity from Tones Comprising Full Harmonic Series. *Journal of the Acoustical Society of America (JASA)*, 124(1):589–597, July 2008. ISSN 1520-8524. doi: 10.1121/1.2931957. URL http://www.ncbi.nlm.nih.gov/pubmed/18647001. 00010.

23 MIDI Manufacturers Association. Complete MIDI 1.0 Detailed Specification V96.1, 2nd edition. Standard, MMA, 2001.

24 Ermanno Briner. *Musikinstrumentenführer*. Philipp Reclam jun., Stuttgart, 3rd edition, 1998.

25 William A Sethares. *Tuning, Timbre, Spectrum, Scale*. Springer, London, 2005. ISBN 978-1-84628-113-6 1-84628-113-X 1-85233-797-4 978-1-85233-797-1. URL http://public.eblib.com/EBLPublic/PublicView.do?ptiID=303730.

26 Carl E Seashore. *Psychology of Music*. McGraw-Hill, New York, 1938.

27 Max Schoen. *The Psychology of Music*. The Ronald Press Company, New York, 1940.

28 Neville H Fletcher. Acoustical Correlates of Flute Performance Technique. *Journal of the Acoustical Society of America (JASA)*, 57(1):233–237, 1975. ISSN 00014966. doi: 10.1121/1.380430. URL http://link.aip.org/link/JASMAN/v57/i1/p233/s1&Agg=doi.

29 Neville Horner Fletcher and Thomas D Rossing. *The Physics of Musical Instruments*. Springer, 2nd edition, 1998. ISBN 978-0-387-98374-5. 01508.

30 Geoffroy Peeters. Music Pitch Representation by Periodicity Measures Based on Combined Temporal and Spectral Representations. In *Proceedings of the International Conference on Acoustics Speech and Signal Processing (ICASSP)*, volume 5, May 2006. doi: 10.1109/ICASSP.2006.1661210.

31 Lawrence R Rabiner and Ronald W Schafer. *Digital Processing of Speech Signals*. Prentice Hall, New Jersey, 1978. ISBN 0-13-213603-1.

32 Anssi P Klapuri. Multiple Fundamental Frequency Estimation Based on Harmonicity and Spectral Smoothness. *Transactions on Speech and Audio Processing*, 11(6): 804–816, November 2003. ISSN 1063-6676. doi: 10.1109/TSA.2003.815516. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1255467.

33 Alfredo Esquivel Jaramillo, Jesper Kjær Nielsen, and Mads Græsbøll Christensen. A Study on How Pre-whitening Influences Fundamental Frequency Estimation. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6495–6499, May 2019. doi: 10.1109/ICASSP.2019.8683653. ISSN: 2379-190X.

34 John Makhoul. Linear Prediction: A Tutorial Review. *Proceedings of the IEEE*, 63(4): 561–580, April 1975. ISSN 1558-2256. doi: 10.1109/PROC.1975.9792. Conference Name: Proceedings of the IEEE.

35 Peter D Welch. The Use of Fast Fourier Transform for the Estimation of Power Spectra: A Method based on Time Averaging over Short, Modified Periodograms. *IEEE Transactions on Audio and Electroacoustics*, 15(2):70–73, June 1967. ISSN 1558-2582. doi: 10.1109/TAU.1967.1161901. Conference Name: IEEE Transactions on Audio and Electroacoustics.

36 Axel Röbel and Xavier Rodet. Efficient Spectral Envelope Estimation and its Application to Pitch Shifting and Envelope Preservation. In *Proceedins of DAFx*, 2005. URL http://recherche.ircam.fr/equipes/analyse-synthese/roebel/paper/trueenv_dafx2005.pdf.

37 Myron J Ross, Harry L Shaffer, Andrew Cohen, Richard Freudberg, and Harold J Manley. Average Magnitude Difference Function Pitch Extractor. *Transactions on Acoustics, Speech, and Signal Processing*, 22(5):353–362, October 1974. ISSN 0096-3518. doi: 10.1109/TASSP.1974.1162598. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1162598.

38 Alain De Cheveigné and Hideki Kawahara. YIN, A Fundamental Frequency Estimator for Speech and Music. *Journal of the Acoustical Society of America (JASA)*, 111(4):1917–1930, 2002. ISSN 00014966. doi: 10.1121/1.1458024. URL http://link.aip.org/link/JASMAN/v111/i4/p1917/s1&Agg=doi.

39 Tetsuya Shimamura and Hajime Kobayashi. Weighted Autocorrelation for Pitch Extraction of Noisy Speech. *Transactions on Speech and Audio Processing*, 9(7):727–730, 2001. ISSN 10636676. doi: 10.1109/89.952490. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=952490.

40 M R Schroeder. Period Histogram and Product Spectrum: New Methods for Fundamental-Frequency Measurement. *Journal of the Acoustical Society of America (JASA)*, 43(4):829, 1968. ISSN 00014966. doi: 10.1121/1.1910902. URL http://link.aip.org/link/JASMAN/v43/i4/p829/s1&Agg=doi. 00226.

41 A Michael Noll. Pitch Determination of Human Speech by the Harmonic Product Spectrum, the Harmonic Sum Spectrum, and a Maximum Likelihood

Estimate. In *Proceedings of the Symposium on Computer Processing in Communications*, volume 19, pages 779–797, Brooklyn, 1969. Polytechnic Press of the University of Brooklyn.

42 A Michael Noll. Short-Time Spectrum and “Cepstrum” Techniques for Vocal-Pitch Detection. *Journal of the Acoustical Society of America (JASA)*, 36(2):296, 1964. ISSN 00014966. doi: 10.1121/1.1918949. URL http://link.aip.org/link/JASMAN/v36/i2/p296/s1&Agg=doi.

43 David C Rife and Robert R Boorstyn. Single Tone Parameter Estimation from Discrete-Time Observations. *IEEE Transactions on Information Theory*, 20(5):591–598, September 1974. ISSN 1557-9654. doi: 10.1109/TIT.1974.1055282. Conference Name: IEEE Transactions on Information Theory.

44 Boris Doval and Xavier Rodet. Fundamental Frequency Estimation Using a New Harmonic Matching Method. In *Proceedings of the International Computer Music Conference (ICMC)*, Montreal, 1991.

45 Ray Meddis and Lowel O’Mard. A Unitary Model of Pitch Perception. *Journal of the Acoustical Society of America (JASA)*, 102(3):1811–20, September 1997. ISSN 0001-4966. URL http://www.ncbi.nlm.nih.gov/pubmed/9714929.

46 Anssi P Klapuri. Auditory Model-Based Methods for Multiple Fundamental Frequency Estimation. In Anssi P Klapuri and Manuel Davy, editors, *Signal Processing Methods for Music Transcription*, pages 229–265. Springer-Verlag, Berlin, 2006.

47 Masataka Goto. A Robust Predominant-F0 Estimation Method for Real-Time Detection of Melody and Bass Lines in CD Recordings. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 2, pages II757–II760 vol.2, June 2000. doi: 10.1109/ICASSP.2000.859070. ISSN: 1520-6149.

48 Chris Chafe, David Jaffe, Kyle Kashima, Bernard Mont-Reynaud, and Julius O Smith. Techniques for Note Identification in Polyphonic Music. In *Proceedings of the International Computer Music Conference (ICMC)*. ICMA, 1985.

49 Thomas W. Parsons. Separation of Speech from Interfering Speech by Means of Harmonic Selection. *Journal of the Acoustical Society of America (JASA)*, 60(4): 911, 1976. ISSN 00014966. doi: 10.1121/1.381172. URL http://link.aip.org/link/JASMAN/v60/i4/p911/s1&Agg=doi. 00257.

50 Anssi P Klapuri, Tuomas Virtanen, and Jan-Markus Holm. Robust Multipitch Estimation for the Analysis and Manipulation of Polyphonic Musical Signals. In *Proceedings of the Conference on Digital Audio Effects (DAFX)*, Verona, 2000.

51 Anssi P Klapuri. Multipitch Estimation and Sound Separation by the Spectral Smoothness Principle. *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing. Proceedings (ICASSP)*, pages 3381–3384,

2001. doi: 10.1109/ICASSP.2001.940384. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=940384.

52 Alain De Cheveigné and Hideki Kawahara. Multiple Period Estimation and Pitch Perception Model. *Speech Communication*, 27:175–185, 1999.

53 Alain De Cheveigné. Multiple F0 Estimation. In DeLiang Wang and Guy J Brown, editors, *Computational Auditory Scene Analysis*, pages 45–79. IEEE Press/Wiley, New Jersey, 2006 00000.

54 T. Miwa, Y. Tadokoro, and T. Saito. Musical pitch estimation and discrimination of musical instruments using comb filters for transcription. In *42nd Midwest Symposium on Circuits and Systems (Cat. No.99CH36356)*, volume 1, pages 105–108 vol. 1, August 1999. doi: 10.1109/MWSCAS.1999.867220.

55 Ray Meddis and Michael J Hewitt. Modeling the identification of concurrent vowels with different fundamental frequencies. *Journal of the Acoustical Society of America (JASA)*, 91(1): 233–245, January 1992. ISSN 0001-4966. doi: 10.1121/1.402767. URL http://www.ncbi.nlm.nih.gov/pubmed/1737874.

56 Anssi P Klapuri. A Perceptually Motivated Multiple-F0 Estimation Method. In *Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, New Paltz, 2005.

57 Matti Karjalainen and Tero Tolonen. Multi-pitch and periodicity analysis model for sound separation and auditory scene analysis. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 929–932 vol. 2, Phoenix, 1999. IEEE. ISBN 0-7803-5041-3. doi: 10.1109/ICASSP.1999.759824. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=759824.

58 Tero Tolonen and Matti Karjalainen. A Computationally Efficient Multipitch Analysis Model. *Transactions on Speech and Audio Processing*, 8(6):708–716, 2000.

59 Daniel D Lee and H Sebastian Seung. Algorithms for Non-negative Matrix Factorization. In T K Leen, T G Dietterich, and V Tresp, editors, *Proceedings of Neural Information Processing Systems (NIPS)*, pages 556–562. MIT Press, 2001. URL http://papers.nips.cc/paper/1861-algorithms-for-non-negative-matrix-factorization.pdf.

60 Andrzej Cichocki, Rafal Zdunek, Anh Huy Phan, and Shun-ichi Amari. *Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-way Data Analysis and Blind Source Separation*. John Wiley & Sons, 2009. ISBN 978-0-470-74666-0. URL https://dl.acm.org/doi/abs/10.5555/1822971.

61 Paris Smaragdis and Judith C Brown. Non-Negative Matrix Factorization for Polyphonic Music Transcription. In *Proceedings of the Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA))*, New Paltz, 2003.

IEEE. ISBN 0-7803-7850-4. doi: 10.1109/ASPAA.2003.1285860. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1285860.

62 Arshia Cont. Realtime Audio to Score Alignment for Polyphonic Music Instruments using Sparse Non-Negative Constraints and Hierarchical HMMs. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 5, Toulouse, 2006.

63 Arnaud Dessein, Arshia Cont, and Guillaume Lemaitre. Real-time Polyphonic Music Transcription with Non-negative Matrix Factorization and Beta-Divergence. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 489–494, Utrecht, Netherlands, 2010. URL https://hal.inria.fr/hal-00708682.

64 Cédric Févotte, Nancy Bertin, and Jean-Louis Durrieu. Nonnegative Matrix Factorization with the Itakura-Saito Divergence: With Application to Music Analysis. *Neural Computation*, 21(3): 793–830, 2009. ISSN 0899-7667. doi: 10.1162/neco.2008.04-08-771. URL https://doi.org/10.1162/neco.2008.04-08-771. Publisher: MIT Press.

65 Paris Smaragdis, Bhiksha Raj, Paris Smaragdis, and Bhiksha Raj. Shift-Invariant Probabilistic Latent Component Analysis, 2008.

66 Nancy Bertin, Roland Badeau, and Emmanuel Vincent. Enforcing Harmonicity and Smoothness in Bayesian Non-Negative Matrix Factorization Applied to Polyphonic Music Transcription. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(3):538–549, March 2010. ISSN 1558-7924. doi: 10.1109/TASL.2010.2041381.

67 Rodrigo Schramm, Andrew McLeod, Mark Steedman, and Emmanouil Benetos. Multi-pitch Detection and Voice Assignment for a cappella Recordings of Multiple Singers. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Suzhou, China, 2017. ISMIR. URL https://qmro.qmul.ac.uk/xmlui/handle/123456789/25292. Accepted: 2017-08-17T10:13:45Z.

68 Tuomas Virtanen. Monaural Sound Source Separation by Nonnegative Matrix Factorization With Temporal Continuity and Sparseness Criteria. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(3): 1066–1074, March 2007. ISSN 1558-7924. doi: 10.1109/TASL.2006.885253. Conference Name: IEEE Transactions on Audio, Speech, and Language Processing.

69 Minje Kim, Jiho Yoo, Kyeongok Kang, and Seungjin Choi. Nonnegative Matrix Partial Co-Factorization for Spectral and Temporal Drum Source Separation. *IEEE Journal of Selected Topics in Signal Processing*, 5(6):1192–1204, October 2011. ISSN 1941-0484. doi: 10.1109/JSTSP.2011.2158803. Conference Name: IEEE Journal of Selected Topics in Signal Processing.

70 Emad M Grais and Hakan Erdogan. Single Channel Speech Music Separation Using Nonnegative Matrix Factorization and Spectral Masks. In *Proceedings* *of the International Conference on Digital Signal Processing (DSP)*, pages 1–6, July 2011. doi: 10.1109/ICDSP.2011.6004924. ISSN: 2165-3577.

71 Paris Smaragdis, Cedric Fevotte, Gautham J Mysore, Nasser Mohammadiha, and Matthew Hoffman. Static and Dynamic Source Separation Using Non-negative Factorizations: A unified view. *IEEE Signal Processing Magazine*, 31(3):66–75, May 2014. ISSN 1053-5888. doi: 10.1109/MSP.2013.2297715. URL http://ieeexplore.ieee.org/search/freesearchresult.jsp?reload=true&queryText=10.1109%2FMSP.2013.2297715.

72 Emmanouil Benetos, Margarita Kotti, and Constantine Kotropoulos. Musical Instrument Classification Using Non-negative Matrix Factorization Algorithms. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2006. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1692967.

73 Emmanouil Benetos, Margarita Kotti, and Constantine Kotropoulos. Application of Non-negative Matrix Factorization to Musical Instrument Classification. In *Proceedings of the International Symposium on Communications, Control and Signal Processing*, Marrakesh, 2006. URL http://users.uoi.gr/cs01702/MargaritaKotti/MypublicationsPDFs/application%20classification.pdf.

74 David S Alves, Jouni Paulus, and José Fonseca. Drum Transcription from Multichannel Recordings with Non-Negative Matrix Factorization. In *Proceedings of the European Signal Processing Conference (EUSIPCO)*, Glasgow, 2009. URL http://www.researchgate.net/publication/229044064_Drum_transcription_from_multichannel_recordings_with_non-negative_matrix_factorization/file/60b7d52c563d962996.pdf.

75 Chih-Wei Wu and Alexander Lerch. Drum Transcription using Partially Fixed Non-Negative Matrix Factorization With Template Adaptation. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Malaga, 2015. ISMIR. doi: 10.5281/zenodo.1417839. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2015/10/Wu_Lerch_2015_Drum-Transcription-using-Partially-Fixed-Non-Negative-Matrix-Factorization-With.pdf.

76 Florian Kaiser and Thomas Sikora. Music Structure Discovery in Popular Music using Non-negative Matrix Factorization. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 429–434, Utrecht, Netherlands, 2010. URL https://pdfs.semanticscholar.org/885d/60fff6de71c562d35564dfacbf4a44e8d1e8.pdf.

77 Ron J Weiss and Juan Pablo Bello. Identifying Repeated Patterns in Music Using Sparse Convolutive Non-Negative Matrix Factorization. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Utrecht, August 2010.

78 Oriol Nieto and Tristan Jehan. Convex Non-negative Matrix Factorization for Automatic Music Structure Identification. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 236–240, May 2013. doi: 10.1109/ICASSP.2013.6637644.

79 Siddharth Gururani and Alexander Lerch. Automatic Sample Detection in Polyphonic Music. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Suzhou, 2017. International Society for Music Information Retrieval (ISMIR). doi: 10.5281/zenodo.1418331. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2017/07/Gururani_Lerch_2017_Automatic-Sample-Detection-in-Polyphonic-Music.pdf.

80 Hirokazu Kameoka, Takuya Nishimoto, and Shigeki Sagayama. A Multipitch Analyzer Based on Harmonic Temporal Structured Clustering. *Transactions on Audio, Speech and Language Processing*, 15(3):982–994, March 2007. ISSN 1558-7916. doi: 10.1109/TASL.2006.885248. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4100686.

81 Matija Marolt. A Connectionist Approach to Automatic Transcription of Polyphonic Piano Music. *IEEE Transactions on Multimedia*, 6(3):439–449, June 2004. ISSN 1941-0077. doi: 10.1109/TMM.2004.827507. Conference Name: IEEE Transactions on Multimedia.

82 Sebastian Böck and Markus Schedl. Polyphonic Piano Note Transcription with Recurrent Neural Networks. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 121–124, Kyoto, Japan, March 2012. doi: 10.1109/ICASSP.2012.6287832. ISSN: 2379-190X.

83 Rainer Kelz, Matthias Dorfer, Filip Korzeniowski, Sebastian Böck, Andreas Arzt, and Gerhard Widmer. On the Potential of Simple Framewise Approaches to Piano Transcription. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, New York, December 2016. URL http://arxiv.org/abs/1612.05153.

84 Curtis Hawthorne, Erich Elsen, Jialin Song, Adam Roberts, Ian Simon, Colin Raffel, Jesse Engel, Sageev Oore, and Douglas Eck. Onsets and Frames: Dual-Objective Piano Transcription. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Paris, June 2018. URL http://arxiv.org/abs/1710.11153.

85 Curtis Hawthorne, Ian Simon, Rigel Swavely, Ethan Manilow, and Jesse Engel. Sequence-to-Sequence Piano Transcription with Transformers. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Online, July 2021. URL http://arxiv.org/abs/2107.09142.

86 Emmanouil Benetos, Simon Dixon, Zhiyao Duan, and Sebastian Ewert. Automatic Music Transcription: An Overview. *IEEE Signal Processing Magazine*, 36(1):20–30, January 2019. ISSN 1558-0792. doi: 10.1109/MSP.2018.2869928.

87 Emilia Gómez and Jordi Bonada. Towards Computer-Assisted Flamenco Transcription: An Experimental Comparison of Automatic Transcription Algorithms as Applied to A Cappella Singing. *Computer Music Journal*, 37(2):73–90, June 2013. ISSN 0148-9267. URL https://doi.org/10.1162/COMJ_a_00180.

88 John Thickstun, Zaid Harchaoui, and Sham M Kakade. Learning Features of Music From Scratch. In *Proceedings of the International Conference on Learning Representations (ICLR)*, page 14, Toulon, France, 2017.

89 Justin Salamon, Emilia Gomez, Daniel P W Ellis, and Gael Richard. Melody Extraction from Polyphonic Music Signals: Approaches, Applications, and Challenges. *IEEE Signal Processing Magazine*, 31(2):118–134, March 2014. ISSN 1558-0792. doi: 10.1109/MSP.2013.2271648. Conference Name: IEEE Signal Processing Magazine.

90 Andrew McLeod and Mark Steedman. Evaluating Automatic Polyphonic Music Transcription. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 8, Paris, 2018.

91 Graham E Poliner and Daniel P W Ellis. A Discriminative Model for Polyphonic Piano Transcription. *EURASIP Journal on Advances in Signal Processing*, 2007 (1):048317, December 2006. ISSN 1687-6180. doi: 10.1155/2007/48317. URL https://doi.org/10.1155/2007/48317.

92 Rachel Bittner and Juan J Bosch. Generalized Metrics for Single-F0 Estimation Evaluation. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 8, Delft, Netherlands, 2019.

93 Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset. In *Proceedings of the International Conference on Learning Representations (ICLR)*, New Orleans, LA, January 2019. URL http://arxiv.org/abs/1810.12247.

94 Helena Cuesta, Emilia Gómez, Agustín Martorell, and Felipe Loáiciga. Analysis of Intonation in Unison Choir singing. In *Proceedings of the International Conference on Music Perception and Cognition (ICMPC)*, Graz, Austria, 2018. Zenodo. doi: 10.5281/ZENODO.1286570. URL https://zenodo.org/record/1286570. Type: dataset.

95 Qingyang Xi, Rachel M Bittner, Johan Pauwels, Xuzhou Ye, and Juan P Bello. GUITARSET: A Dataset for Guitar Transcription. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 8, Paris, France, 2018.

96 ISO 16:1975. Acoustics – Standard tuning frequency (Standard musical pitch). Standard 16:1975, ISO, 1975.

97 Yongweil Zhu, Mohan S Kankanhalli, and Sheng Gao. Music Key Detection for Musical Audio. In *Proceedings of the 11th International Multimedia Modelling Conference*, Melbourne, 2005.

98 Alexander Lerch. On the Requirement of Automatic Tuning Frequency Estimation. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Victoria, 2006. ISMIR. doi: 10.5281/zenodo.1414812. URL http://dx.doi.org/10.14279/depositonce-2037.

99 Yi Qin and Alexander Lerch. Tuning Frequency Dependency in Music Classification. In *Proceedings of the International Conference on Acoustics Speech and Signal Processing (ICASSP)*, pages 401–405, Brighton, UK, 2019. Institute of Electrical and Electronics Engineers (IEEE). doi: 10.1109/ICASSP.2019.8683340. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2019/04/Qin-and-Lerch-2019-Tuning-Frequency-Dependency-in-Music-Classificatio.pdf.

100 Alexander Lerch. Ein Ansatz zur automatischen Erkennung der Tonart in Musikdateien. In *Proceedings of the VDT International Audio Convention (23. Tonmeistertagung)*, Leipzig, November 2004. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2016/10/Lerch-2004-Ein-Ansatz-zur-automatischen-Erkennung-der-Tonart-.pdf.

101 Martin Riedmiller and Heinrich Braun. A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. In *Proceedings of the International Conference on Neural Networks*, San Francisco, 1993. IEEE. doi: 10.1109/ICNN.1993.298623.

102 Eric D Scheirer. Using Musical Knowledge to Extract Expressive Performance Information from Audio Recordings. In *Proceedings?*, 1995.

103 Simon Dixon. A Dynamic Modelling Approach to Music Recognition. In *Proceedings of the International Computer Music Conference (ICMC)*, Hong Kong, August 1996.

104 Simon Dixon, Dan Tidhar, and Emmanouil Benetos. The Temperament Police: The Truth, the Ground Truth, and Nothing but the Truth. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 7, Miami, FL, 2011.

105 Sebastian Ewert and Meinard Müller. Chroma Toolbox: Matlab Implementations for Extracting Variants of Chroma-Based Audio Features. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 6, Miami, FL, 2011.

106 Geoffroy Peeters. Musical Key Estimation of Audio Signal Based on Hidden Markov Modeling of Chroma Vectors. In *Proceedings of the International Conference on Digital Audio Effects (DAFX)*, Montreal, Quebec, Canada, 2006.

107 Matti P Ryynänen. Probabilistic Modelling of Note Events in the Transcription of Monophonic Melodies. Master's thesis, Tampere University of Technology, 2004.

108 Karin Dressler and Sebastian Streich. Tuning Frequency Estimation using Circular Statistics. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Wien, September 2007.

109 Alessio Degani, Marco Dalai, Riccardo Leonardi, and Pierangelo Migliorati. Comparison of Tuning Frequency Estimation Methods. *Multimedia Tools and Applications*, 74(15):5917–5934, 2014. ISSN 1573-7721. doi: 10.1007/s11042-014-1897-2. URL https://doi.org/10.1007/s11042-014-1897-2.

110 Carol L Krumhansl. *Cognitive Foundations of Musical Pitch*. Oxford University Press, New York, 1990.

111 Takuya Fujishima. Realtime Chord Recognition of Musical Sound: a System Using Common Lisp Music. In *Proceedings of the International Computer Music Conference (ICMC)*, 1999.

112 Mark A Bartsch and Gregory H Wakefield. To Catch a Chorus: Using Chroma-Based Representations for Audio Thumbnailing. In *Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, New Paltz, 2001. IEEE.

113 George Tzanetakis, Andrey Ermolinskyi, and Perry Cook. Pitch Histograms in Audio and Symbolic Music Information Retrieval. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Paris, 2002.

114 Hendrik Purwins, Benjamin Blankertz, and Klaus Obermayer. A New Method for Tracking Modulations in Tonal Music in Audio Data Format. In *IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN'00)*, volume 6, pages 6270–6275, Como, 2000.

115 Markus Cremer and Claas Derboven. A System for Harmonic Analysis of Polyphonic Music. In *Proceedings of the 25th International Audio Engineering Society Conference: Metadata for Audio*, June 2004.

116 Alan V Oppenheim, Donald H Johnson, and K Steiglitz. Computation of Spectra with Unequal Resolution using the Fast Fourier Transform. *Proceedings of the IEEE*, 59(2): 299–301, 1971. ISSN 0018-9219. doi: 10.1109/PROC.1971.8146. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1450076.

117 Holger Kirchhoff and Alexander Lerch. Evaluation of Features for Audio-to-Audio Alignment. *Journal of New Music Research*, 40(1):27–41, 2011. doi: 10.1080/09298215.2010.529917. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2015/04/Kirchhoff_Lerch_2011_Evaluation%20of%20Features%20for%20Audio-to-Audio%20Alignment.pdf.

118 Filip Korzeniowski and Gerhard Widmer. Feature Learning for Chord Recognition: The Deep Chroma Extractor. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, New York, December 2016. URL http://arxiv.org/abs/1612.05065.

119 Hélène Papadopoulos and Geoffroy Peeters. Large-scale Study of Chord Estimation Algorithms based on Chroma Representation and HMM. In *Proceedings of the International Workshop on Content-Based Multimedia Indexing (CBMI)*, Bordeaux, 2007.

120 Jan Weil and Jean-Louis Durrieu. An HMM-based Audio Chord Detection System Attenuating the Main Melody. In *Proceedings of the Music Information Retrieval EXchange (MIREX) at the 9th International Conference on Music Information Retrieval (ISMIR)*, Philadelphia, PA, September 2008.

121 Johan Pauwels, Matthias Varewyck, and Jean-Pierre Martens. Audio Chord Extraction using a Probabilistic Model. In *Proceedings of the Music Information Retrieval EXchange (MIREX) at the 9th International Conference on Music Information Retrieval (ISMIR)*, Philadelphia, September 2008.

122 Juan Pablo Bello and Jeremy Pickens. A Robust Mid-level Representation for Harmonic Content in Music Signals. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, London, September 2005.

123 Christopher A Harte, Mark B Sandler, and Martin Gasser. Detecting Harmonic Change in Musical Audio. In *Proceedings of the 1st ACM Workshop on Audio and Music Computing Multimedia (AMCMM)*, Santa Barbara, 2006. ACM. 00088.

124 Nanzhu Jiang, Peter Grosche, Verena Konz, and Meinard Muller. Analyzing Chroma Feature Types for Automated Chord Recognition. page 11, 2011.

125 Steffen Pauws. Musical key extraction from audio. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Barcelona, 2004.

126 Matthias Varewyck, Johan Pauwels, and Jean-Pierre Martens. A Novel Chroma Representation of Polyphonic Music Based on Multiple Pitch Tracking Techniques. *Proceedings of the 16th ACM International Conference on Multimedia (MM)*, page 667, 2008. doi: 10.1145/1459359.1459455. URL http://portal.acm.org/citation.cfm?doid=1459359.1459455.

127 Matti P Ryynänen and Anssi P Klapuri. Automatic Transcription of Melody, Bass Line, and Chords in Polyphonic Music. *Computer Music Journal*, 32(3):72–86, September 2008. ISSN 0148-9267. doi: 10.1162/comj.2008.32.3.72. URL http://www.mitpressjournals.org/doi/abs/10.1162/comj.2008.32.3.72.

128 Özgür Izmirli. Template based key finding from audio. In *Proceedings of the International Computer Music Conference (ICMC)*, Barcelona, September 2005.

129 Filip Korzeniowski and Gerhard Widmer. End-to-end Musical Key Estimation using a Convolutional Neural Network. In *Proceedings of the European Signal Processing Conference (EUSIPCO)*, pages 966–970, August 2017. doi: 10.23919/EUSIPCO.2017.8081351.

130 Hendrik Schreiber and Meinard Müller. Musical Tempo and Key Estimation using Convolutional Neural Networks with Directional Filters. In *Proceedings of the Sound and Music Computing Conference (SMC)*, Malaga, Spain, March 2019. URL http://arxiv.org/abs/1903.10839.

131 David Temperley. The Tonal Properties of Pitch-Class Sets : Tonal Implication, Tonal Ambiguity, and Tonalness. *Computing in Musicology*, 15:24–38, 2007.

132 Cian O'Brien and Alexander Lerch. Genre-Specific Key Profiles. In *Proceedings of the International Computer Music Conference (ICMC)*, Denton, 2015. ICMA. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2015/09/O\stquoteBrien_Lerch_2015_Genre-Specific%20Key%20Profiles.pdf.

133 Andrew J Milne, William A Sethares, Robin Laney, and David B Sharp. Modelling the Similarity of Pitch Collections with Expectation Tensors. *Journal of Mathematics and Music*, 5(1):1–20, March 2011. ISSN 1745-9737. doi: 10.1080/17459737.2011.573678. URL http://www.tandfonline.com/doi/abs/10.1080/17459737.2011.573678.

134 Elaine Chew. *Towards a Mathematical Model of Tonality*. PhD thesis Dissertation, Massachusetts Institute of Technology, 1998.

135 Ching-Hua Chuan and Elaine Chew. Polyphonic Audio Key Finding Using the Spiral Array CEG Algorithm. In *International Conference on Multimedia and Expo*, pages 21–24. IEEE, 2005. ISBN 0-7803-9331-7. doi: 10.1109/ICME.2005.1521350. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1521350.

136 Gabriel Gatzsche, Markus Mehnert, and Christian Stöcklmeier. Interaction with tonal pitch spaces. In *Proceedings of the 8th International Conference on New Interfraces for Musical Expression (NIME)*, Genova, 2008.

137 Peter Knees, Angel Faraldo, Perfecto Herrera, Richard Vogl, Sebastian Böck, Florian Horschlager, and Mickael Le Goff. Two Data Sets for Tempo Estimation and Key Detection in Electronic Dance Music Annotated from User Corrections. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 364–370, 2015. URL http://dblp.uni-trier.de/search?q=two%20data%20sets%20for%20tempo.

138 Johan Pauwels, Ken O'Hanlon, Emilia Gomez, and Mark B Sandler. 20 Years of Automatic Chord Recognition from Audio. In *Proceedings of the 20th Conference of the International Society for Music Information (ISMIR)*, Delft, The Netherlands, November 2019. URL https://zenodo.org/record/3527739#.YBcfd3dKg8M.

139 Taemin Cho and Juan P Bello. On the Relative Importance of Individual Components of Chord Recognition Systems. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(2): 477–492, February 2014. ISSN 2329-9304. doi: 10.1109/TASLP.2013.2295926. Conference Name: IEEE/ACM Transactions on Audio, Speech, and Language Processing.

140 Eric J Humphrey and Juan Pablo Bello. Rethinking Automatic Chord Recognition with Convolutional Neural Networks. In *Proceedings of the International Conference on Machine Learning and Applications (ICMLA)*, volume 2, pages 357–362, December 2012. doi: 10.1109/ICMLA.2012.220.

141 Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Audio Chord Recognition with Recurrent Neural Networks. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Curitiba, 2013. ISMIR. URL http://www-etud.iro.umontreal.ca/ boulanni/ISMIR2013.pdf.

142 Xinquan Zhou and Alexander Lerch. Chord Detection Using Deep Learning. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Malaga, 2015. ISMIR. doi: 10.5281/zenodo.1416968. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2015/10/Zhou_Lerch_2015_Chord-Detection-Using-Deep-Learning.pdf.

143 Filip Korzeniowski and Gerhard Widmer. A Fully Convolutional Deep Auditory Model for Musical Chord Recognition. In *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6, September 2016. doi: 10.1109/MLSP.2016.7738895.

144 Christon-Ragavan Nadar, Jakob Abeßer, and Sascha Grollmisch. Towards CNN-based Acoustic Modeling of Seventh Chords for Automatic Chord Recognition. In *Proceedings of the Sound and Music Computing Conference (SMC)*, page 7, Malaga, Spain, 2019.

145 Arun Shenoy and Ye Wang. Key, Chord, and Rhythm Tracking of Popular Music Recordings. *Computer Music Journal*, 29(3):75–86, September 2005. ISSN 0148-9267. doi: 10.1162/0148926054798205. URL http://www.mitpressjournals.org/doi/abs/10.1162/0148926054798205.

146 Alexander Sheh and Daniel P W Ellis. Chord Segmentation and Recognition using EM-Trained Hidden Markov Models. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Baltimore, MD, 2003. 00231.

147 Laurent Oudre, Cédric Fevotte, and Yves Grenier. Probabilistic Template-Based Chord Recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(8): 2249–2259, November 2011. ISSN 1558-7924. doi: 10.1109/TASL.2010.2098870.

148 Andrew J Viterbi. Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. *Transactions on Information Theory*, 13:260–269, 1967.

149 Kyogu Lee and Malcolm Slaney. Automatic Chord Recognition from Audio Using a Supervised HMM Trained with Audio-from-Symbolic Data. In *Proceedings of the 1st ACM workshop on Audio and music computing multimedia (AMCMM)*, Santa Barbara, 2006. ACM. 00019.

150 Hendrik Vincent Koops, W Bas de Haas, John Ashley Burgoyne, Jeroen Bransen, Anna Kent-Muller, and Anja Volk. Annotator Subjectivity in Harmony Annotations of Popular Music. *Journal of New Music Research*, 48(3):232–252, May 2019. ISSN 0929-8215. doi: 10.1080/09298215.2019.1613436. URL https://doi.org/10.1080/09298215.2019.1613436. Publisher: Routledge _eprint: https://doi.org/10.1080/09298215.2019.1613436.

151 Johan Pauwels and Geoffroy Peeters. Evaluating Automatically Estimated Chord Sequences. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 749–753, Vancouver, BC, Canada, May 2013. doi: 10.1109/ICASSP.2013.6637748. ISSN: 2379-190X.

152 John Ashley Burgoyne, Jonathan Wild, and Ichiro Fujinaga. An Expert Ground-Truth Set for Audio Chord Recognition and Music Analysis. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 6, Miami, FL, 2011.

153 Colin Raffel, Brian McFee, Eric J Humphrey, Justin Salamon, Oriol Nieto, Dawen Liang, and Daniel P W Ellis. mir_eval: A Transparent Implementation of Common MIR Metrics. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 6, Taipei, Taiwan, 2014.

154 Adam Berenzweig, Beth Logan, Daniel P W Ellis, and Brian Whitman. A Large-Scale Evaluation of Acoustic and Subjective Music-Similarity Measures. *Computer Music Journal*, 28(2):63–76, June 2004. ISSN 0148-9267. doi: 10.1162/014892604323112257. URL https://doi.org/10.1162/014892604323112257.

155 Trevor de Clercq and David Temperley. A Corpus Analysis of Rock Harmony. *Popular Music*, 30(1):47–70, January 2011. ISSN 0261-1430, 1474-0095. doi: 10.1017/S026114301000067X. URL https://www.cambridge.org/core/product/identifier/S026114301000067X/type/journal_article.

156 Masataka Goto. RWC Music Database: Popular, Classical, and Jazz Music Databases. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 2, Paris, 2002.

157 Matthias Dorfer, Jan Haji? Jr, Andreas Arzt, Harald Frostel, and Gerhard Widmer. Learning Audio–Sheet Music Correspondences for Cross-Modal Retrieval and Piece Identification. *Transactions of the International Society for Music Information Retrieval*, 1(1):22–33, September 2018. ISSN 2514-3298. doi: 10.5334/tismir.12. URL http://transactions.ismir.net/articles/10.5334/tismir.12/. Number: 1 Publisher: Ubiquity Press.

8

Intensity

Intensity, magnitude, and loudness-related features constitute one of the most commonly used classes for the description of audio content. Most audio editors and digital audio workstations illustrate the audio signal in its waveform view, its amplitude variation over time. There also exists a variety of instruments for level, volume, and loudness measurements frequently used in recording studio environments.

8.1 Human Perception of Intensity and Loudness

It is important to distinguish the meaning of the terms *intensity* and *loudness*. Intensity means a physical, measurable entity such as the magnitude of a sound while loudness refers to a perceptual entity that can only be measured via responses of human observers [1]. Unfortunately, the term *loudness* is also used for *algorithmic models* of the perceived loudness.

Human perception of intensity is related to the magnitude of the audio signal in a way that if the signal's magnitude is scaled up, the perceived loudness will increase as well. It has been discovered very early that this relationship is nonlinear; a linear increase in the signal's magnitude or power will not result in a linear increase in perceived loudness. An approximately linear relation could be found by using the pseudo-unit *decibel* (dB) which is computed by taking the logarithm of the intensity feature $v(n)$ (computed from a block of samples, see below):

$$v_{\mathrm{dB}}(n) = 20 \cdot \log_{10}\left(\frac{v(n)}{v_0}\right) = 10 \cdot \log_{10}\left(\frac{v^2(n)}{v_0^2}\right) \tag{8.1}$$

with v_0 representing a reference constant. In the digital domain, dealing with audio amplitudes in the range of $[-1, 1]$, it is commonly set to $v_0 = 1$. The resulting

An Introduction to Audio Content Analysis: Music Information Retrieval Tasks & Applications,
Second Edition. Alexander Lerch.
© 2023 The Institute of Electrical and Electronics Engineers, Inc. Published 2023 by John Wiley & Sons, Inc.

level unit is then referred to as dBFS (*decibel full scale*) and has a range of $-\infty \leq v_{\mathrm{dB}}(n) \leq 0$ dB.[1] The scaling factor before the logarithm has been chosen to scale the nonlinear function so that 1 dB roughly represents the level difference a human can easily recognize. This is only a rough approximation as the actual so-called *Just Noticeable Difference in Level (JNDL)* depends on the stimulus level and partly on stimulus frequency and masking effects [2].

As the input magnitudes decrease to silence, the logarithm decreases to $-\infty$. To avoid this (and very low values for inputs close to 0) in practical implementations, two approaches are common. First, a threshold ϵ can be defined so that all inputs lower than G_{T} will be set to ϵ:

$$v_{\mathrm{trunc}}(n) = \begin{cases} v(n), & \text{if } v(n) \geq G_{\mathrm{T}} \\ G_{\mathrm{T}}, & \text{otherwise.} \end{cases} \tag{8.2}$$

This approach comes at the cost of an additional `if` statement per feature value.

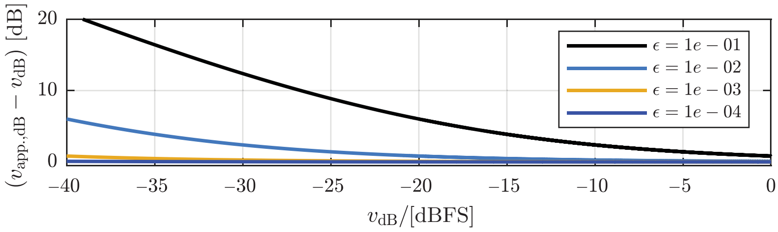
Second, the calculation of log(0) can be avoided by adding a small constant ϵ:

$$v_{\mathrm{approx,dB}}(n) = 20 \cdot \log_{10}(v(n) + \epsilon). \tag{8.3}$$

The choice of ϵ determines the measurement accuracy at low-level inputs. The measurement error increases for decreasing $v(n)$ approaching ϵ. It will be 6 dB for $v(n) = \epsilon$ and increase with lower levels. Figure 8.1 visualizes the error amount for different ϵ.

The decibel scale is not a loudness scale since equal-sized steps on the decibel scale are not perceived as equal-sized loudness steps by human listeners: doubling the level in dB does not result in doubling the perceived loudness of a sound. Stevens, summarizing several loudness perception studies, proposed a simple rule of thumb stating that a doubling of the perceived loudness roughly corresponds to a level increase of 10 dB [3].

As has been shown by several studies throughout the last century, models of the human perception of loudness that take both the input signal's frequency and



Repo 1: plotLogEpsilon.m

Figure 8.1 Level error introduced by adding a small constant ϵ to the argument of a logarithm dependent on the input level v_{dB}.

---

[1] Note that higher dBFS levels are possible if the implementation allows for them.

the cochlea's frequency resolution into account tend to be more accurate. The most important researchers in the history of loudness perception are probably Fletcher and Munson [4, 5], Stevens [1, 3], Moore and Glasberg [6], and Zwicker and Fastl [2].

8.2 Representation of Dynamics in Music

In a traditional musical score, loudness-related performance instructions are rather vague. Usually only five to eight different dynamic steps are used to describe musical dynamics (e.g. ***pp***: *pianissimo*, ***p***: *piano*, ***mf***: *mezzoforte*, ***f***: *forte*, ***ff***: *fortissimo* for the dynamic range from "very soft" to "very strong"), complemented by indications of smooth loudness transitions (e.g. *crescendo* or *decrescendo* for increasing and decreasing loudness, respectively) and dynamic accents (e.g. ***sf***: *sforzando*). These written instructions do not directly refer to absolute loudness as the actual loudness also depends on a number of other influencing factors such as instrumentation, timbre, number of voices, performance, and musical tension and musical context. The difference between musical dynamics and absolute loudness can also be illustrated by the fact that while listening to a recording on a hi-fi system, the reproduction volume may be manipulated without losing the *piano* or *forte* character of the performance. Still, Nakamura has shown that measures of intensity or loudness can, to a certain degree, be used as indication of musical dynamics [7]. A loudness-related performance attribute is the *tremolo*, the periodic modulation of loudness over time. It often appears in combination with a vibrato.

A more technical representation of dynamics in music is the *velocity* as standardized in the musical instrument digital interface (MIDI) protocol [8]. It consists of 128 volume steps with the highest representing maximal intensity. But although the number of velocity steps is standardized, there is no standardized relationship between MIDI velocity and intensity: Goebl and Bresin found that for the Yamaha Disklavier and the Bösendorfer SE System (both pianos allowing for the monitoring of performance data), the relationship between MIDI velocity and (logarithmic) sound pressure level is nearly linear when disregarding very low and high values [9]. Dannenberg investigated the root mean square (RMS) peak level of various synthesizers and software instruments and found great differences among different synthesizers [10]. He identified a general trend for the velocity to be related to the square root of the RMS peak instead of its logarithm. Using one single electronic instrument, Taguti measured the A-weighted sound pressure level dependent on velocity and key [11]. The results, displayed over various keys for different input velocities, showed nonsystematic deviations of up to 10 dB from a constant level among keys.

8.3 Features

Many of the intensity features presented here are instantaneous features similar to the features introduced in Chapter 3.5. Therefore, the same postprocessing options (e.g. computing derived features, aggregate features within texture windows, normalization, see Section 3.7) can be applied to most of these features. The features presented in this section can be roughly structured into three categories: physical measures of sound intensity, approaches to measure intensity in recording studio environments, and psycho-acoustically motivated features modeling the human perception of loudness.

8.3.1 Root Mean Square

The root mean square (*RMS*) is one of the most common intensity features and is sometimes directly referred to as the sound intensity. It is calculated from a block of audio samples by

$$v_{\mathrm{RMS}}(n) = \sqrt{\frac{1}{\mathcal{K}} \sum_{i=i_{\mathrm{s}}(n)}^{i_{\mathrm{e}}(n)} x(i)^2}. \tag{8.4}$$

Note that for blocks with zero mean, the RMS resembles the variance (see Section 3.1.3.4). Typical block lengths for the RMS calculation are in the range of several hundred milliseconds. This length is also referred to as the *integration time.*

The result of the calculation is a value within the range $0 \leq v_{\mathrm{RMS}}(n) \leq 1$ (as long as the input amplitude 1 represents full scale). It will equal 0 if the input is silence and will approach 1 for both a square wave with maximum amplitude and a constant DC offset at ±1. Sharp transients in the signal will be smoothed out by an RMS measure due to the comparably long integration time. The RMS is often reported in dBFS.

The calculation of the RMS can be computationally inefficient for large block lengths and small hop sizes $\mathcal{H}$. If the hop size is one sample, it is possible to reduce the number of computations by using the method of recursive implementation introduced in the context of the moving average (MA) filter in Appendix A.2.6.1:

$$v_{\mathrm{RMS}}^2(n) = \frac{x(i_{\mathrm{e}}(n))^2 - x(i_{\mathrm{s}}(n-1))^2}{i_{\mathrm{e}}(n) - i_{\mathrm{s}}(n) + 1} + v_{\mathrm{RMS}}^2(n-1), \tag{8.5}$$

$$v_{\mathrm{RMS}}(n) = \sqrt{v_{\mathrm{RMS}}^2(n)}. \tag{8.6}$$

This implementation is computationally efficient, but it still requires a significant amount of memory to be allocated for large block lengths. An approximation of the RMS v^*_{RMS} with hop size $\mathcal{H} = 1$ can be implemented with a single-pole filter (compare Appendix A.2.6.2):

$$v_{\mathrm{tmp}}(i) = \alpha \cdot v_{\mathrm{tmp}}(i-1) + (1-\alpha) \cdot x(i)^2 \tag{8.7}$$

$$v^*_{\mathrm{RMS}}(i) = \sqrt{v_{\mathrm{tmp}}(i)}. \tag{8.8}$$

The filter coefficient α can be estimated from the integration time (or block length) with Eq. (16.21). Listing 8.1 shows the implementation of a blocked RMS implementation with a single-pole filter.

```
% single pole implementation
v_sp = filter(alpha, [1 -(1-alpha)], x.^2);

% per block standard implementation
for n = 1:size(x_b, 1)
    i_start = (n-1) * iHopLength + 1;
    i_stop = min(length(x), i_start + iBlockLength - 1);

    % calculate the rms
    vrms(1, n) = sqrt(mean(x_b(n, :).^2));
    vrms(2, n) = max(sqrt(v_sp(i_start:i_stop)));
end

% convert to dB
epsilon = 1e-5; %-100dB
vrms(vrms < epsilon) = epsilon;
vrms = 20*log10(vrms);
```

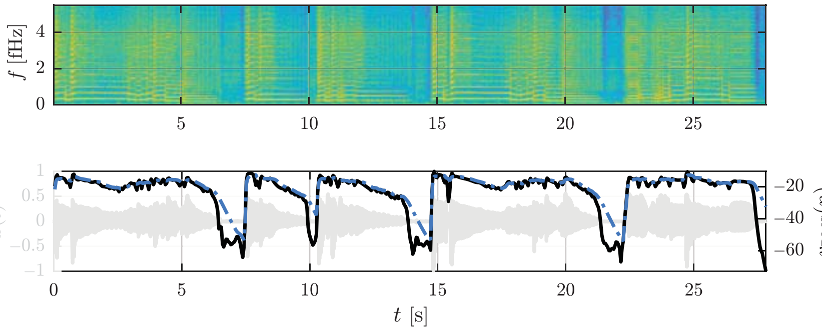
Repo 2: FeatureTimeRms.m
Repo 3: FeatureTimeRms.py

Listing 8.1 Implementation of *rms* from the time signal $x(i)$.

Figure 8.2 shows the RMS of an example signal for the two implementations described by Eqs. (8.4) and (8.7). The RMS is a measure of the power of the signal; the two implementations shown are roughly equivalent except during sudden signal pauses in which the low-pass filtered variant only slowly decreases.

8.3.2 Weighted Root Mean Square

In many scenarios, the RMS computation is preceded by a weighting filter to simulate the frequency sensitivity of the human ear. This means the audio signal is filtered before computing the RMS as described above (see Figure 8.3). The transfer function of a weighting filter is usually modeled after an inverse *equal-loudness contour* which measures the level for which a listener perceives equal loudness at different frequencies [4]. Thus, the weighting filter amplifies frequency regions in which the human ear is sensitive and attenuates other regions.



Repo 1: plotFeatureTimeRms.m

Figure 8.2 Spectrogram (top), waveform (bottom background), and RMS output (bottom foreground) of a saxophone signal.

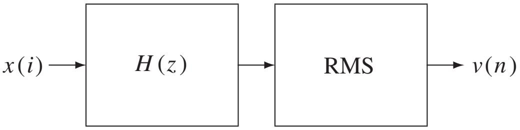
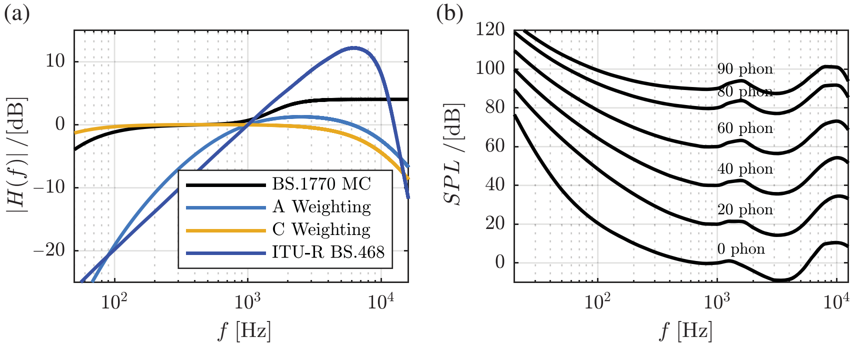


Figure 8.3 Flowchart of the frequency-weighted RMS calculation.

Common transfer functions of the weighting filter are the following:

- *A weighting*: weighting function to be used for sounds at low-level [12],
- *C weighting*: weighting function to be used for sounds at medium-level [12],
- *RLB weighting*: weighting function according to ITU-R BS.1770 (plus high-frequency emphasis for multichannel signals) [13], and
- *CCIR weighting*: weighting function according to ITU-R BS.468 [14].

Figure 8.4 displays the listed transfer functions on the left (a); the right plot shows the equal-loudness contours for comparison. Note the different shape of the contours for different levels. When frequency weighted RMS measures are used



Repo 1: plotLoudnessWeighting.m

Figure 8.4 (a) Frequency weighting transfer functions applied before RMS measurement, (b) equal loudness contours. Source: Based on Ref. [15].

as models of loudness, the integration time is usually several seconds in order to ignore short-term variations of the signal.

8.3.3 Peak Envelope

The *peak envelope* of an audio signal can be extracted in different ways. The simplest way of extracting the envelope is to find the absolute maximum per block of audio samples:

$$v_{\text{Peak}}(n) = \max_{i_s(n) \le i \le i_e(n)} |x(i)|. \tag{8.9}$$

The envelope can also be extracted on a sample-by-sample basis with a so-called *Peak Program Meter (PPM)*. The PPM, frequently used in recording studio environments, operates with different integration times for attack (*attack time*) and release states (*release time*). Typically, the attack time is significantly shorter than the release time (e.g. attack time ≈ 10 ms, release time ≈ 1500 ms) which means that the output reflects an increase in level faster than a decrease. This originates in requirements of recording engineers: the attack time has to be short in order to allow the systems to detect short peaks, while the longer release time gives humans more time to actually see those peaks.

The structure of a digital PPM as described by Zölzer is shown in Figure 8.5 [16].

The filter coefficient representing the attack time is named α_{AT}. The release time coefficient α_{RT} is not always in use and is represented by λ in the figure. More specifically, λ has the following two states depending on the input magnitude:

$$\lambda = \begin{cases} \alpha_{\text{RT}}, & \text{if } |x(i)| < v_{\text{PPM}}(i-1) \\ 0, & \text{otherwise.} \end{cases} \tag{8.10}$$

The two states may be referred to as *release state* and *attack state*. The corresponding output equations are

- *Release state*:

$$\begin{aligned} v_{\text{Peak,PPM}}(i) &= v_{\text{PPM}}(i-1) - \alpha_{\text{RT}} \cdot v_{\text{PPM}}(i-1) \\ &= (1-\alpha_{\text{RT}}) \cdot v_{\text{PPM}}(i-1), \end{aligned} \tag{8.11}$$

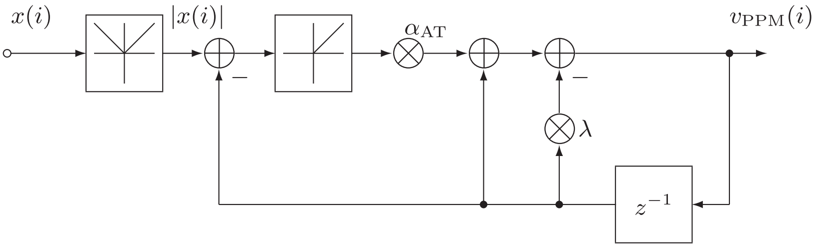


Figure 8.5 Flowchart of a Peak program meter.

- *Attack state*:

$$\begin{aligned} v_{\text{Peak,PPM}}(i) &= \alpha_{\text{AT}} \cdot \big(|x(i)| - v_{\text{PPM}}(i-1)\big) + v_{\text{PPM}}(i-1) \\ &= \alpha_{\text{AT}} \cdot |x(i)| + (1-\alpha_{\text{AT}}) \cdot v_{\text{PPM}}(i-1). \end{aligned} \tag{8.12}$$

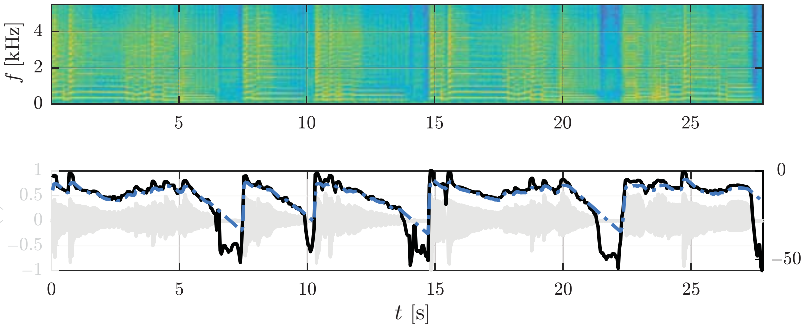
The result of both v_{Peak} and $v_{\text{Peak,PPM}}$ is a value within the range $0 \leq v_{\text{Peak}} \leq 1$. It will equal 0 if the input is silence and approach 1 for certain full-scale signals. Listing 8.2 shows the implementation.

```
for n = 1:iNumOfBlocks
    % calculate the maximum
    vppm(1, n) = max(abs(x_b(n, :)));

    % calculate the PPM value
    v_tmp = ppm_I(x_b(n, :), v_tmp(iHopLength), alpha);
    vppm(2, n) = max(v_tmp);
end

% convert to dB avoiding log(0)
epsilon = 1e-5; %-100dB
vppm(vppm < epsilon) = epsilon;
vppm = 20*log10(vppm);
```

Repo 2: FeatureTimePeakEnvelope.m
Repo 3: FeatureTimePeakEnvelope.py

Listing 8.2 Implementation of *peakenvelope* from the time signal $x(i)$.

Figure 8.6 shows the results of both envelope measures. The comparison with the two RMS results as shown in Figure 8.2 reveals similar behavior of RMS and peak measures as the calculation is relatively similar. The dotted peak maximum shows faster and more pronounced changes as can be clearly seen during the pauses with the PPM's constant decrease due to the long release time.



Repo 1: plotFeatureTimePeakEnvelope.m

Figure 8.6 Spectrogram (top), waveform (bottom background), and PPM output compared to the magnitude maximum per block (bottom foreground, blue and gold, respectively) of a saxophone signal.

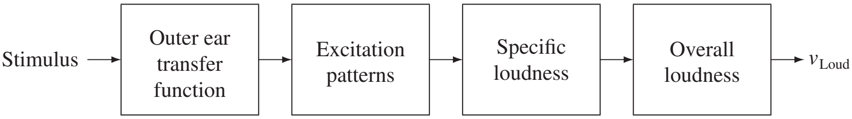


Figure 8.7 Flowchart of Zwicker's model for loudness computation.

8.3.4 Psycho-Acoustic Loudness Features

There exist complex loudness measurements based on either psycho-acoustic properties of human loudness perception or physiological models of the human ear (or both). These are not frequently used for audio content analysis (ACA), as (i) they are comparably costly to implement and to compute, and (ii) there are indications that they are in many cases equally meaningful as simpler loudness approximations such as the algorithm described in *International Telecommunication Union (ITU)* recommendation BS.1770, a weighted RMS solution [17].

A widely known and psycho-acoustically motivated loudness measurement has been proposed by Zwicker and Fastl [2, 18, 19]. Figure 8.7 presents the flowchart of this loudness calculation. The signal is transformed onto the nonlinear frequency axis of the bark domain (see Section 7.1) by a filterbank or a similar frequency transform. The so-called *excitation patterns* are computed from the filterbank outputs by taking into account frequency sensitivity and masking effects. Then, the *specific loudness* is calculated from the excitation patterns per band. The resulting loudness is the sum of the specific loudness over all bands.

One relatively recent recommendation for measurement of the loudness of a program or a file has been published by the *European Broadcasting Union (EBU)* [20]. The loudness itself is measured in compliance to ITU recommendation BS.1770, an RMS measure with an *RLB weighting*. The required block length is 3.0 s with a block overlap ratio of at least 66%. Blocks with very low loudness are discarded with a gating threshold. The EBU recommendation also requires the following additional values to be extracted from the audio signal:

- clipped values according to an up-sampled true peak meter, and
- the loudness range computed from a histogram of all loudness block results as the difference $Q_v(0.95) - Q_v(0.10)$.

8.4 Exercises

8.4.1 Questions

1. Given a signal magnitude of 1, what is its level in dBFS?
2. When implementing the computation of a level in decibel, discuss solutions for dealing with an input signal equaling 0.

8.4.2 Assignments

1. Peak Envelope
   (a) Implement a function `[v_peak] = extract_peak(x_B, fs)` which returns the peak envelope of the signal `x` blocked with function `block_audio` from Section 3.8.2. Return the absolute maximum per block in dBFS. If you haven't implemented this function from a previous assignment yet, follow the following instructions:
      - Implement a function `[fs, x] = read_audio(cPath2Audio)` which reads a wave audio file with scipy.io.wavfile.read and returns the sample rate `fs` and a numpy array `x` with dimensions `numSamples`× `numChannels`. Ensure that the maximum possible value range of `x` is always $[-1, 1[$ regardless of the wordlength of the input file. Support the following wordlengths/input types: `uint8`, `int16`, `int32`, `float32`
      - Implement a function `[x_m] = downmix_audio(x)` which takes a numpy array `x` with dimensions `numSamples` × `numChannels` and returns the downmixed numpy array `x_m` with dimensions `numSamples` × 1.
      - Implement a function `[x_B, t_B] = block_audio(x_m, blockSize, hopSize, fs)` which returns a numpy array `x_B` (dimension `N` × `blockSize`) and a vector t_B (dimension `N` × 1) for blocking the input audio signal into overlapping blocks. Make sure that each sample of the audio signal is included in at least one of the `N` blocks. `t_B` will refer to the start time of each block in seconds.

   (b) Implement a function `[v_ppm] = extract_ppm(x_m, t_attack_InS, t_release_InS, fs)` that implements the PPM. Note that in this case, the output `v_ppm` will have the same length as the input `x_m`. Return the output in dBFS.

   (c) Plot `v_ppm` and compare it with the waveform of the audio signal. How does the feature change when you change attack and/or release time?
2. RMS
   (a) Generate an audio signal with a sample rate of 48 kHz consisting of uniformly distributed white noise of length 10 s.

   (b) Implement a function `[x_filtered] = apply_biquad(x)` with a second-order Infinite Impulse Response (IIR) filter and set the following coefficients as defined for the 48 kHz weighting curve from the ITU recommendation BS.1770:

```
a_1 = -1.99004745483398
a_2 = 0.99007225036621
b_0 = 1.0
b_1 = -2.0
b_2 = 1.0
```

(c) Implement a function `[v_rms] = extract_rms(x_B)` (in dBFS) (see also Section 3.8.2) to compute the RMS of your noise signal. The function returns a numpy RMS vector (in dBFS) with its length equaling the number of blocks in `x_B`. Compute the output for the input noise signal. Then, apply the filter to the noise signal and extract the RMS of the signal filtered with the weighting filter. Discuss the differences.

References

1 Stanley Smith Stevens. *Hearing - Its Psychology and Physiology*. American Institute of Physics for the Acoustical Society of America, New York, 1983 reprint edition, 1938.

2 Eberhard Zwicker and Hugo Fastl. *Psychoacoustics. Facts and Models*. Springer, 2nd edition, 1999. ISBN 978-3-540-23159-2.

3 Stanley Smith Stevens. The Measurement of Loudness. *Journal of the Acoustical Society of America (JASA)*, 27(5):815–829, 1955. 00392.

4 Harvey Fletcher and W A Munson. Loudness, Its Definition, Measurement and Calculation. *Journal of the Acoustical Society of America (JASA)*, 5(2):82, 1933. ISSN 00014966. doi: 10.1121/1.1915637. URL http://link.aip.org/link/JASMAN/v5/i2/p82/s1&Agg=doi.

5 Harvey Fletcher. Auditory Patterns. *Reviews of Modern Physics*, 12(1):47–65, 1940.

6 Brian C J Moore and Brian R Glasberg. Suggested Formulae for Calculating Auditory-Filter Bandwidths and Excitation Patterns. *Journal of the Acoustical Society of America (JASA)*, 74(3):750–753, 1983.

7 Toshie Nakamura. The Communication of Dynamics Between Musicians and Listeners Through Musical Performance. *Perception & Psychophysics*, 41(6):525–533, 1987.

8 MIDI Manufacturers Association. *Complete MIDI 1.0 Detailed Specification V96.1*, 2nd edition. Standard, MMA, 2001.

9 Werner Goebl and Roberto Bresin. Measurement and Reproduction Accuracy of Computer-Controlled Grand Pianos. *Journal of the Acoustical Society of America (JASA)*, 114(4):2273–2283, 2003. doi: 10.1121/1.1605387.

10 Roger B Dannenberg. The Interpretation of MIDI Velocity. In *Proceedings of the International Computer Music Conference (ICMC)*, New Orleans, November 2006.

11 Tomoyasu Taguti. Mapping a Physical Correlate of Loudness into the Velocity Space of MIDI-Controlled Piano Tones. In *Proceedings of the Stockholm Music Acoustics Conference (SMAC)*, Stockholm, August 2003.

12 IEC 61672:2002. Electroacoustics – Sound Level Meters – Part1: Specifications. Standard 61672:2002, IEC, 2002. 00000.

13 ITU-R. BS.1770:2006. Algorithms to Measure Audio Programme Loudness and True-Peak Audio Level. Recommendation, ITU, 2006.

14 ITU-R. BS.468:1986. Measurement of Audio-Frequency Noise Voltage Level in Sound. Recommendation, ITU, 1986.

15 ISO. ISO 226:2003. Technical Report ISO 226:2003, 2003. URL https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/03/42/34222.html.

16 Udo Zölzer. *Digital Audio Signal Processing*. John Wiley & Sons Ltd, Stuttgart, 2nd edition, 2008. ISBN 978-0-470-99785-7.

17 Gilbert A Soulodre and Scott G Norcross. Objective Measures of Loudness. In *Proceedings of the 115th Audio Engineering Society Convention (Preprint No. 5896)*, New York, 2003. Audio Engineering Society.

18 DIN 45631:1991. Berechnung des Lautstärkepegels und der Lautheit aus dem Geräuschspektrum. Standard 45631:1991, DIN, 1991.

19 ISO 532-1:2017. ISO 532-1:2017. Technical report, 2017. URL https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/06/30/63077.html.

20 EBU. R 128: Loudness Normalisation and Permitted Maximum Level of Audio Signals. Technical Report August, EBU, 2010.

9

Temporal Analysis

The temporal aspects of music signals such as the tempo and the rhythm include important musical properties. For example, a simple sequence of pitches does not make a melody without temporal, structured patterns grouping the pitches [1]. This chapter focuses on extracting temporal properties from musical audio, starting with the onset, the beginning of a musical sound event such as a tone or a stroke on a percussive instrument, over the tempo and meter to high-level structural properties of music.

9.1 Human Perception of Temporal Events

During the process of human perception, the audio stream will be segmented into a series of events. Speaking of segmentation is a simplification because musical meaning and even rhythm can be conveyed by audio streams with no such clear division into distinct events [2]. However, this simplification can be assumed to be sufficiently valid in the context of Western music for the majority of possible input signals – other incarnations of temporal information will be ignored for the sake of simplicity.

9.1.1 Onsets

As stated above, an *onset* is the start of a (musical) sound event. The start time of an event is usually considered to be more important than the time of the end of that event, as listeners apparently tend to perceive musical events in terms of Inter-Onset Intervals IOIs [3]. The term onset is frequently used synonymously to onset time, but it would be more correct to state that its time position (i.e. the onset time) is one – probably the main – property of the onset while it can have other properties such as its strength.

An Introduction to Audio Content Analysis: Music Information Retrieval Tasks & Applications,
Second Edition. Alexander Lerch.
© 2023 The Institute of Electrical and Electronics Engineers, Inc. Published 2023 by John Wiley & Sons, Inc.

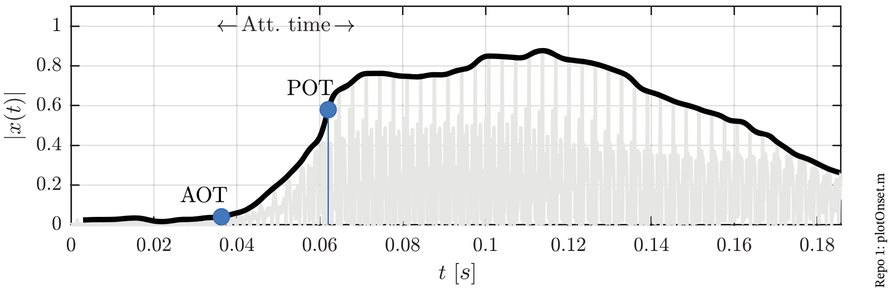


Figure 9.1 Visualization of an envelope, attack time, and possible location of the perceptual onset time (POT).

The onset time, however, is not easily pinpointed either. In reality, the start of a musical sound usually is not an exact point in time, but a time span. This time span is referred to as the *attack time* or *initial transient time*, starting with the first instrument-induced oscillation and ending if the maximum amplitude is reached or the sound reaches its quasiperiodic state. An example of an attack phase of a note is shown in Figure 9.1.

The attack time varies considerably between different musical instruments, groups of instruments, and playing techniques. It ranges from about 5 ms for some percussive instruments to up to 200 ms for woodwind instruments (flute) under certain conditions [4].

The terms *onset*, *attack*, and *transient* are often used inconsistently and interchangeably, which complicates exact communication. Only the context gives insights into whether these terms have the same or different meanings, or whether they stand for a signal state, a time span, or a point in time. To give an example of a different naming convention than the one used here, Bello et al. propose to use the terms *attack* instead of attack time, *transient* as a description of the initial phase of a musical event in which "the signal evolves quickly in some nontrivial or relatively unpredictable way" (a period covering our attack time but extending until a quasiperiod state is reached), and *onset* as a single instant chosen to mark the temporally extended transient (instead of onset time) [5].

Repp pointed out that three definitions of onset times can generally be distinguished [6]:

1. *Note Onset Time (NOT)*: the time when the instrument is triggered to make a sound. In the MIDI domain, the NOT is exactly the time of the Note-On command. Depending on the instrument or sample used for sound generation, this is not necessarily the time when the signal becomes detectable or audible.
2. *Acoustic Onset Time (AOT)*: the first time when a signal or an acoustic event is theoretically measurable. Sometimes the AOT is called *physical onset time*.

3. *Perceptual Onset Time (POT)*: the first time when the event can be perceived by the listener. The POT might further be distinguished from the *Perceptual Attack Time (PAT)* which is the instant of time that is relevant for the perception of rhythmic patterns [7]. While the PAT might occur later than the POT, they will be equal in many cases. For the sake of simplicity, there will be no distinction of POT and PAT in the following.

The POT can never occur before the AOT, which in turn never occurs before the NOT. Due to the "perceptual" definition of the POT, the exact location cannot be determined acoustically but has to be measured in a listening test. Both Gordon and Zwicker found strong location drifts of the PAT depending on the waveform properties during the rise time [7, 8]. There are indications of the POT being correlated with the envelope slope [7].

Given the three definitions above, the following question arises: which of the three onset times is, on the one hand, detectable in the signal and, on the other hand, of the utmost interest in automatic onset detection and rhythm-related audio analysis tasks? Due to the symbolic nature of the NOT, it simply cannot be detected from the audio signal. The choice between AOT and POT might be application-dependent; assuming that musicians adapt their timing to their sound perception and that most audio content analysis (ACA) systems try to analyze the *perceptible* audio content, the POT is most likely the point in time desired as result. This reflection, however, is rather academic since, in reality, the time accuracy of automatic onset detection systems is too poor to differentiate between the onset time definitions for all but a small class of signal types. Furthermore, the time accuracy of ground-truth data for testing practically limits what accuracy can be tested for.

As these data are annotated by humans, the human ability to locate onset times and to distinguish closely spaced onsets limits the expected ground truth accuracy. Furthermore, since most ACA systems aim to be at least as accurate as human perception, this perceptual detection accuracy also allows useful insights into a reasonable time accuracy of an onset detection system. In the following, a few noteworthy studies on the perception of onset times are summarized to clarify the time ranges under consideration.

Hirsh found that temporal discrimination of two onsets is possible for humans if the onset time difference is as little as 2 ms [9]. However, in order to determine the order of the stimuli, their distance had to be about 20 ms. The measurements were done with synthetic signals with short rise times.

Gordon reported a standard deviation of 12 ms for the accuracy of onset times specified by test listeners, using 16 real-world monophonic sounds of different instruments played in an infinitely long loop pattern with *IOIs* of 600 ms [7]. Friberg and Sundberg undertook a similar experiment using tone stimuli [10].

For IOIs smaller than 240 ms, they reported a just noticeable difference of about 10 ms and increasing values for larger IOIs.

Repp reported for the manual annotation of onset times in piano recordings a mean absolute measurement error of about 4.3 ms and a maximum error of about 35 ms [11]. Leveau et al. compared the results of three test subjects annotating the onset times in audio files of various genres and instrumentations [12]. The results showed a mean absolute measurement error over all test data of about 10 ms; for one piece of classical music, the mean absolute measurement error nearly reached 30 ms.

Rasch evaluated the onset time differences between instruments in three ensemble performances [13]. He found synchronization deviations in a range between 30 and 50 ms between the (string and woodwind) instruments, while the mean onset time differences were in the range of ±6 ms. However, since the measurement accuracy has not been evaluated in this case, it is unknown how much of the actual time differences can be attributed to the performance itself.

For piano duet performance, Shaffer reported standard deviations within the voices between 14 and 38 ms [14].

It may be concluded that the accuracy of human onset perception depends on the test data and that deviations evoked by motoric abilities seem to be in the same range. The presented results imply that an automatic onset detection system aiming at human detection accuracy (or being evaluated with test data annotated by humans) will have a mean error no lower than 5–10 ms; the error can be expected to be as much as 10 times higher for specific instruments and pitches with long rise times.

9.1.2 Tempo and Meter

The *tempo* determines the “pace” of a musical piece [15]. It is the rate of occurrence of perceived periodic events (*beats*) at regular intervals [16], with these intervals occurring at a moderate and natural rate [17]. This perceived tempo is called the *tactus* [18] and is sometimes simply referred to as the *foot tapping rate* [19, 20]. The natural rate of this tactus is usually in the general range of 1–3 Hz, corresponding to 60–180 BPM [21, 22]. It is important to note that the tactus is a perceptual concept: while it is determined by groups and accents of note events, the pulse of a tactus may or may not fall on a note event and a note event may or may not fall on a pulse.

For segments of music with constant tempo, the tempo $\mathfrak{T}$ in Beats per Minute (BPM) can be computed using the length of the segment Δt_s in seconds and the number of beats $\mathcal{B}$ in the segment:

$$\mathfrak{T} = \frac{\mathcal{B} \cdot 60\ \mathrm{s}}{\Delta t_\mathrm{s}} \quad [\mathrm{BPM}]. \tag{9.1}$$

In the case of a dynamic tempo, the local tempo can be extracted by identifying the event time of every beat t_b and computing the distance between two neighboring beats with indices j and $j+1$:

$$\mathfrak{T}_\mathrm{local}(j) = \frac{60\ \mathrm{s}}{t_\mathrm{b}(j+1) - t_\mathrm{b}(j)}\quad [\mathrm{BPM}]. \tag{9.2}$$

The interbeat distance can also be preprocessed by a low-pass filter to smooth out potential inaccuracies in measuring the beat times.

Assigning a single, *overall* tempo to a piece of music is difficult when the tempo is not constant; in this case, the mean tempo given in Eq. (9.1) does not necessarily match the *perceived overall tempo* a listener would indicate. This led Gabrielsson to distinguishing between the mean tempo and the main tempo, the latter being a measure ignoring slow beginnings or final *ritardandi* [23]. Repp found good correlation of the perceived tempo with the mean value of a logarithmic IOI distribution [24]. Goebl proposed a mode tempo which is computed by sweeping a window over the histogram of *Inter-Beat Intervals IBIs* and selecting the maximum position as predominant tempo [25].

To complicate matters further, McKinney and Moelants argued that a single tempo does not sufficiently describe the (listener) group response when presented with a piece of music. They propose a representation of the overall tempo with two BPM values instead of a single one [26].

The *meter* is a regular alternation of strong and weak musical elements which are grouped with a length of normally three to seven beats or a length of around 3–5 s.

The metrical structure of music is typically hierarchical with higher-hierarchical levels grouping events in lower-hierarchical levels. Figure 9.2 visualizes different groupings depending on the time signature for two bars. Note that different time signatures might lead to different tempi even if the shortest event length, the *tatum*, is identical.

9.1.3 Rhythm

The *rhythm* is specified by a pattern of a grouped series of events and durations [16]. The grouping properties allow a hierarchical segmentation into smaller subsequences forming different grouping levels. The length of the groups can range from a few notes up to whole parts of the work, defining musical form [18].[1] Groups of a length between one beat and the length of the meter are most commonly referred to as rhythm. The rhythm is then defined by its event times, note and rest durations, and accents; if the durations of subsequent intervals relate

---

[1] However, in this text, the term *rhythm* will be used only for groups with a length of up to several beats.

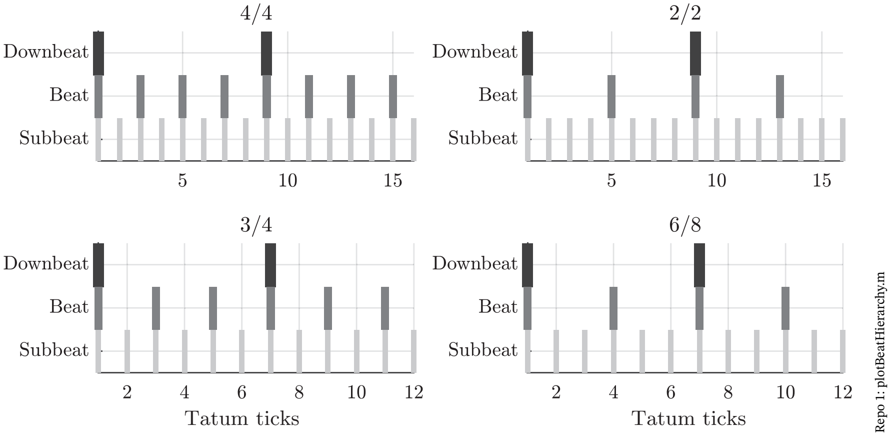


Figure 9.2 Different hierarchical levels related to tempo, beat, and meter.

to simple integer ratios, then the group usually has a closer binding than otherwise [27].

The various hierarchical levels of temporal grouping are an important property of many (Western) pieces of music. Humans perceive pulses at different levels and with different tempi; at all levels, the grouping of strong and weaker events occurs. The basic building block on the lowest (and shortest) level is commonly referred to as tatum [28], although other terms such as *atomic beat* have been used [29]. The tatum specifies the lowest period length or the period of the regular pulse train with the highest frequency represented in the music. Every rhythm is built of the tatums which can be interpreted as a rhythmic grid or a time quantization. The length of the highest-level grouping depends on the definition of grouping and could go up to the level defining musical form such as the length of musical phrases or even longer structures which form groups.

9.1.4 Timing

The *timing* of individual notes or temporal events in a music performance does not necessarily exactly reflect the notated structural properties of the rhythm or meter but shows (minor) temporal deviations from the underlying rhythmic structure [30], often to convey structural properties of the music or to add expressiveness to the performance. Expressive timing will also be discussed in Chapter 15.

9.2 Representation of Temporal Events in Music

The representation of musical (temporal) events in the musical score is closely related to the perception of such events.

9.2.1 Tempo and Time Signature

Although the target tempo is sometimes specified in the musical score, it is ultimately chosen by the performing artists. Tempo instructions for the performers became more and more explicit over the centuries. While many pieces from the Baroque period do not contain instructions due to the composer's assumption that the tempo was specified by performance conventions, tempo indications started to occur more frequently in later epochs; Italian terms such as *Largo* (very slow), *Adagio* (slow), *Andante* (walking pace), *Moderato* (moderately), *Allegro* (fast), and *Presto* (very fast) are commonly used. The last century has seen more specific tempo indications in BPM.

The *local tempo* varies throughout a piece of music for nearly all genres. The possibilities to include instructions for such variations in the score are limited besides adding tempo indicators; examples of tempo instructions for sliding tempo changes are *ritardando* (slowing down) and *accelerando* (speeding up).

The *bar* (also called a *measure*) is the score equivalent of the (perceptual) meter. A score marks the beginning and end of each bar by a vertical line. The first beat of a bar usually has the highest (perceptual) weight and is referred to as the *downbeat*.

The *time signature* is a way to convey information on the properties of a bar, namely the number of beats grouped together in one bar (upper part of the time signature) as well as the note value constituting one beat (lower part of the time signature). The time signatures in Figure 9.3 group four, three, two, and two beats, respectively. The fourth example differs from the first three in grouping half notes instead of quarter notes. The "denominator" of the time signature thus indicates the note value of one beat, while its "numerator" indicates the number of beats per bar.[2]

9.2.2 Note Value

The *note value* defines the relative length of a note with respect to time signature and tempo. Notational convention requires that the sum of note values and rest values per bar (except a few special cases) must equal the numerator of the time signature. Thus, the absolute onset time of each note is specified by the bar index and the note's position in the bar, given a specific tempo.

The offset time (also the *note off time*) is determined by the note's onset time and its note value in the score but is not necessarily as clearly defined in a real-world

Figure 9.3 Frequently used time signatures.

---

[2] There are exceptions from this rule such as a time signature $^{6}/_{8}$ with a beat length of three eighth notes.

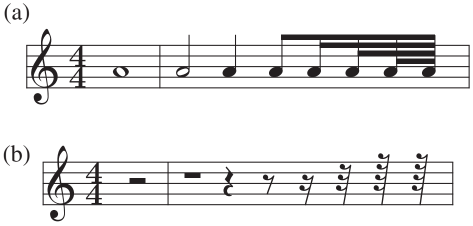


Figure 9.4 Note values (a) and corresponding rest values (b) with decreasing length in musical score notation.

performance. Sometimes a note value is shortened and a rest is appended to give the performing artists indications of the preferred articulation. In general, however, depending on epoch, style, and composer it is not unusual to place the responsibility for such articulation decisions on the performers rather than the musical score.

Figure 9.4 shows the most common note values (a) starting from a whole note and decreasing the value down to two sixty-fourth notes and the corresponding rests (b).

9.3 Onset Detection

The goal of *onset detection* is to segment the audio stream into separate musical events. This supports tasks such as tempo detection or automatic music transcription and enables the analysis of rhythm and timing.

The flowchart of a typical *onset detection* system (also referred to as *onset tracking* system) is shown in Figure 9.5. First, a so-called novelty function is computed. This novelty function represents a measure of the amount of "new" information in the audio signal for each analysis block and thus indicates the instantaneous likelihood of a musical event starting. Second, a *peak picking* process is applied to the novelty function to identify the locations of the significant maxima which can then be regarded as onset times.

Overview articles for different approaches to detecting onsets have been published by Bello et al. and Dixon [5, 31].

9.3.1 Novelty Function

An important property of the beginning of musical sound events is that "something new happens." Thus, the first step toward automatic onset detection is the

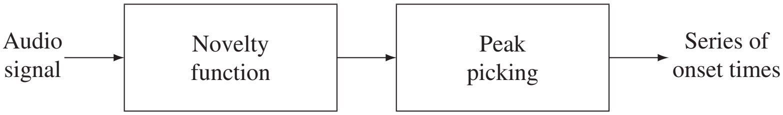


Figure 9.5 General flowchart of an onset detection system.

computation of a *novelty function* which indicates the amount of audio signal changes over time [32]. Other names of this function are *detection function* [5] or *difference function* [33].

The first step in the computation of the *novelty function* is usually the calculation of the difference between current and preceding feature values. The result is then smoothed and negative values are discarded by applying HWR. The latter processing step is usually helpful for onset detection as an (amplitude or energy) increase might be expected at onset times, while a decrease should make an onset less likely.

In one of the first approaches to onset detection in (percussive) music signals, Schloss presented an algorithm that makes direct use of the audio signal's envelope slope, extracted in the time domain [34]. He extracts the envelope of the audio signal by computing the maximum of the magnitude of the signal within a block of samples and recommends to adjust the block length to the length of the period of the lowest frequency present. The envelope slope is then computed by using linear regression over several points of the peak amplitude.

As pointed out in Chapter 8, there exist different possibilities to extract the envelope of a signal, including taking the block's maximum amplitude and low-pass filtering either the signal's peak magnitudes or its RMS. An envelope-based analysis can work well for simple signals (e.g. drum loops); if the input signal is a mixture of multiple simultaneous sound sources, these methods applied to the time-domain signal invariably fail. Although this challenge can be partly addressed by applying the envelope analysis on multiple frequency subbands instead of the broadband audio signal, methods utilizing short time Fourier transform (STFT) inputs have been shown to be generally more reliable. The reason for this is that the STFT analysis allows to take into account spectral differences such as a pitch change in addition to amplitude and envelope changes. The disadvantage of the STFT-based computation is the comparably poor time resolution which affects the algorithm's detection accuracy.

Modern systems generally use deep neural networks (DNNs) to extract the novelty function from a time-frequency input representation. Typical architectures include Recurrent Neural Networks RNNs [35, 36] and convolutional neural networks (CNNs) [37]. Traditional approaches, however, estimate the novelty from the differences between subsequent (overlapping) blocks of audio data, commonly in the frequency domain. This difference can either be computed with each individual spectral bin or with multiple bins grouped into frequency bands.

The frequency resolution of the input representation for onset detection systems increased over time. Early systems used one or a few frequency bands. For example, Scheirer used a filterbank of six bands basically covering a one-octave range [38], Klapuri proposed to use 21 nonoverlapping bands with their bandwidth and midfrequency inspired by Zwicker's critical bands [39], and Duxbury used five bands up to 22 kHz with constant Q [40]. Zhou and Reiss proposed to use a filterbank of first-order complex resonators with an overall number of 960 frequency bands and 10 filters covering the range of a semitone, respectively [41]. Nowadays, virtually all approaches use a STFT input representation or a similar time-frequency representation.

While spectral domain onset detection systems differ in the number of frequency bands they analyze, their main difference is the distance measure $\mathfrak{d}(n)$ between consecutive STFTs. One standard way to compute the distance is the spectral flux (see Section 3.5.8). However, since only magnitude increases as opposed to decreases are of interest (onsets as opposed to offsets), Eq. (3.56) is often modified to discard magnitude decreases by applying a HWR (see Eq. 3.59) before summation:

$$\mathfrak{d}(n) = \frac{\sqrt{\sum\limits_{k=0}^{\mathcal{K}/2} \mathrm{HWR}(|X(k,n)| - |X(k,n-1)|)^2}}{{}^{\mathcal{K}}\!/_{2} + 1}. \tag{9.3}$$

Setting all negative intermediate results to 0 ensures that the novelty function does not confuse onsets and offsets. An implementation of this flux variant is shown in Listing 9.1.

```
% difference spectrum (set first diff to zero)
afDeltaX = diff([X(:, 1), X], 1, 2);

% half-wave rectification
afDeltaX(afDeltaX<0) = 0;

% flux
d_flux = sqrt(sum(afDeltaX.^2)) / size(X, 1);
```

Repo 2: NoveltyFlux.m
Repo 3: NoveltyFlux.py

Listing 9.1 Implementation of the half-wave rectified spectral flux.

Many other variations and/or extensions of this STFT-based distance measure have been proposed. Some apply nonlinear scaling to the magnitude values; Laroche proposed the use of applying a square root function to increase the impact of lower amplitudes [42] and Hainsworth and Macleod calculated a logarithmic distance [33]. It is also possible to compute the distance with the cosine distance between two STFT frames as suggested by Foote [32] or to use the distance between complex STFT bins [43]. Bello et al. pointed out that phase relations may be used for the detection of novelty in an audio stream as well [44]. In particular, they suggest to apply the principles of instantaneous frequency computation (see Appendix B.6) and use the difference of the unwrapped phases.

A common problem of the distance measures above is that they are susceptible to frequency changes that do not mark the start of a musical event such as vibrato or portamento. Goto and Muraoka proposed to address this problem by taking into account not only the differences of magnitudes at the identical bin indices but also differences to neighboring magnitudes in the preceding STFT block [45]. Note that the usefulness of this distance strongly depends on the parametrization, especially the ratio of STFT size and sample rate as well as the block overlap ratio.

Röbel proposed a transient detection that utilizes the center of gravity (COG) of the instantaneous energy per frequency band [46]. The derivation of time reassignment is closely related to frequency reassignment (see Appendix B.6) and is a way of virtually improving the time resolution.

Experience shows that the extracted function $\mathfrak{d}(n)$ often contains too much noise for reliable processing. Therefore, a smoothing filter is commonly applied to the function before further processing. Listing 9.2 shows the implementation of such postprocessing with a Moving Average (MA) filter.

```
% novelty function
d = hNoveltyFunc(X, f_s);

% smooth novelty function
b = ones(iSmoothLpLen,1)/iSmoothLpLen;
d = filtfilt (b, 1, d);
d(d<0) = 0;

% compute threshold
iLen = min(iThreshLpLen, floor(length(d) / 3));
b = ones(iLen, 1) / iLen;
G_T = .4*mean(d(2:end)) + filtfilt (b, 1, d);

[fPeaks, iPeaks] = findpeaks(max(0, d - G_T));
```

Repo 2: computeNoveltyFunction.m
Repo 3: computeNoveltyFunction.py

Listing 9.2 Implementation of smoothing and peak picking the novelty function.

9.3.2 Peak Picking

As the goal of onset detection is the detection of events in time, these discrete time locations have to be extracted from the novelty function. Other downstream tasks such as tempo and rhythm extraction might also work on the novelty function itself instead of the extracted series of onsets.

The onsets are estimated by *peak picking* the novelty function. A perfect novelty function would indicate an onset at each local maximum and would be zero everywhere else. In reality, however, just detecting the locations of local maxima without applying other constraints will likely cause a large number of falsely detected onsets (False Positives (FPs), see Section 6.1.1). To suppress peaks of no interest, a threshold G_d is applied to the novelty function and only local maxima above this threshold are considered as onset time candidates. Practically, all local maxima after subtracting the threshold from the novelty function: $\mathfrak{d}(n) - G_d$ are considered as onsets.

In the simplest case, the threshold is a fixed threshold:

$$G_{d,\mathrm{c}} = \lambda_1. \tag{9.4}$$

A potential problem with the fixed threshold is that the scale and shape of the novelty function can change considerably between different input audio signals. Therefore, an alternative to the fixed threshold is a signal-adaptive threshold. This adaptive threshold could be computed from the smoothed version of the novelty function. A typical smoothing filter is the MA filter:

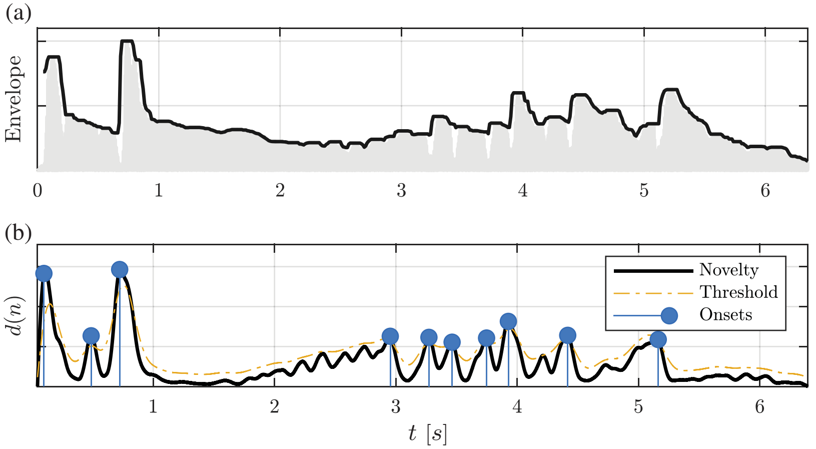
$$G_{d,\mathrm{ma}}(n) = \lambda_2 + \sum_{j=0}^{\mathcal{O}-1} b(j) \cdot d\left(n + \mathcal{O}/2 - j\right) \tag{9.5}$$

with $b(j)$ representing a user-defined window function. The parameter λ_2 shifts the threshold for adjusting the algorithm's sensitivity; low values of λ_2 will generally lead to high recall R, while high λ_2 values will lead to high precision P. Alternatively, a *median filter* may replace the MA filter; its output $\hat{Q}_d(0.5)$ is an estimate of the median in the block of an appropriate length $\mathcal{K}$:

$$G_{d,\mathrm{me}}(n) = \lambda_2 + \hat{Q}_d(0.5). \tag{9.6}$$

Figure 9.6 displays a simple amplitude-based novelty function and adaptive threshold and the detected onset times for the first phrase of the jazz standard "Summertime" played by a solo saxophone.

In addition to the thresholding process, other constraints have been proposed to improve peak picking reliability for onset detection. The confidence for a local maximum being an onset can, for example be indicated by the "amplitude"



Repo 1: plotNovelty.m

Figure 9.6 Audio signal and extracted envelope (a) and novelty function with example threshold and picked onsets (b).

distance between the local maximum and the preceding local minimum or the measurement of the novelty function's slope before the local maximum.

A typical problem with the raw output of untweaked onset detection systems is the occurrence of onsets with very close proximity in time. The reason for this can, on the one hand, be a polyphonic input signal with slightly asynchronous onset times supposed to be played synchronously or, on the other hand, simply a detection error of the system. For many tasks, onsets with a proximity of less than a minimum distance, e.g. 100 ms are not desirable, so that it is beneficial to combine two or more closely neighbored onsets into one. Simple ways to do so include (i) choosing the earliest onset, (ii) choosing the onset with the highest weight or confidence, or to (iii) compute some kind of average for the resulting (combined) onset time.

The final result of the onset detection system is a series of estimated onset times $\hat{t}_0(j)$, possibly including additional properties such as strength or confidence. The markers in Figure 9.6 indicate the estimated onset positions, estimated with the thresholding shown in Listing 9.2.

9.3.3 Evaluation

In order to evaluate an onset detection system, a series of ground truth onset times have to be compared to the series of detected onset times. Besides potential issues with the labeling accuracy of annotators, a typical problem influencing both annotation and detection is the potential asynchrony of supposedly synchronous onsets: for example different voices can be arpeggiated or a melody note might be slightly delayed with respect to the accompaniment for expressive reasons. It is methodically unclear whether these closely spaced onsets should count as individual events or should be combined into one onset, and, if the latter, how that combination should take place.

9.3.3.1 Metrics

The predicted onset times have to be compared with the reference onset times as given by the ground truth. Commonly, classification metrics are used to evaluate onset detection systems. In order to do that, predicted onsets within a tolerance interval around each ground truth interval are considered as correct, while others are incorrect. Typical tolerance intervals range from ±25 ms to ±50 ms.

Two possible errors can occur: a False Negative (FN) indicating that no onset is detected at the time of a reference onset, and a FP which is an onset that is wrongly detected where no reference onset is found (see Section 6.1.1). Note that there are also borderline cases that have to be addressed specifically, such as two predicted onsets within one tolerance window. Also note that this task is not exactly a classification task, as the number of "Negatives" is unknown; thus, there is no way to count the True Negatives (TNs) without discretizing the time axis. This makes it impossible to compute the accuracy; however, precision, recall, and f-measure can still be computed (see Section 6.1.1) and are considered standard metrics for onset detection [35,37,47,48].

Several other measurements of detection performance have been proposed in the past utilizing the total number of detections ($TP + FP$) and the total number of reference onsets ($TP + FN$). For example, Liu et al. proposed [49]:

$$q_{\text{liu}} = \frac{\max\left((TP + FP), (TP + FN)\right) - (FN + FP)}{\max\left((TP + FP), (TP + FN)\right)}, \tag{9.7}$$

however, this measure can – at least theoretically – be negative.

Most of the metrics mentioned above give the same weight to both FPs and FNs. This is not necessarily desirable, as sometimes a higher precision can be preferable over a high recall or vice versa. This can be visualized with the *Receiver Operating Characteristic Curve (ROC)*, where each data point represents a specific parametrization (e.g. the shift of the onset detection threshold) and the True Positive Rate (TPR) is plotted over the False Positive Rate (FPR). It is obvious that the TPR can be expected to increase with the FPR; however, this relation is not necessarily linear. Therefore, identifying the “working point” in the ROC with the desired ratio of TPR to FPR leads back to the desired parametrization.

The evaluation of an onset detection system might also consider the timing accuracy of the system instead of simply matching onsets for evaluation. Such a measure of proximity takes into account the time distance $\Delta t_{\mathrm{o}}(j) = t_{\mathrm{o}}(j) - \hat{t}_{\mathrm{o}}(j)$ between the reference and the detected onset time, which is ideally 0. From the distribution of the resulting time differences, multiple metrics can be computed. First, the arithmetic mean of the time distances

$$d_{\text{mean}} = \sum_{\forall j} \Delta t_{\mathrm{o}}(j) \tag{9.8}$$

indicates the tendency of the system detecting onsets systematically too early or too late. Second, mean absolute error (MAE) and the mean squared error (MSE), see Eqs. (6.6) and (6.7), are measures of the average time distance between detected and ground truth onset. The standard deviation (see Eq. (3.20))

$$\sigma_d = \sqrt{\frac{1}{(TP + FP)} \sum_{\forall j} \left(\Delta t_{\mathrm{o}}(j) - d_{\text{mean}}\right)^2} \tag{9.9}$$

is a measure of how broadly spread out the deviations are, and the absolute maximum deviation

$$d_{\max} = \max_{\forall j} |\Delta t_{\mathrm{o}}(j)| \tag{9.10}$$

gives a worst-case result for the system. Note that deviation-based metrics can only be computed for True Positives (TPs); thus, they are not meant to replace but only complement the classification-based metrics listed above.

9.3.3.2 Datasets

The common datasets for training and evaluating onset detection performance are usually small, as the manual annotation of all onset times is a tedious and time-consuming task [11, 12]. Alternatively, it is possible to use acoustic recordings with a symbolic trigger (such as recordings of the Yamaha Disklavier) or audio data synthesized from symbolic data, with the potential problem that the data might not generalize or not reflect real-world characteristics, respectively.

For these reasons, general datasets are scarce, and many studies fall back on old datasets with known deficiencies [12]. Other datasets are either from monophonic sources [50, 51], only annotate specific onsets such as drum hits [52–55], or restrict the instrumentation to piano [56].

9.3.3.3 Results

Typical errors of onset detection systems are, for example FPs that occur due to amplitude or frequency changes without a new musical event (tremolo and vibrato) or FNs occurring due to other voices masking an onset. Results for state-of-the-art systems largely depend on the complexity of the input data and how optimized the system is for specific input audio characteristics. Thus, the results of state-of-the-art systems have F-Measures ranging from roughly 60–90%.[3]

9.4 Beat Histogram

The *beat histogram* or *beat spectrum* is not an analysis task itself but a representation that captures some rhythmic properties of the signal. However, since it is usually computed from the novelty function introduced in the previous section, it is mentioned in this place.

Similar to the “normal” magnitude spectrum, the x-axis of the beat spectrum is a frequency (in this case with the unit BPM) and its y-axis represents some form of magnitude (beat strength). Peaks in the histogram should therefore correspond to the main tactus and its integer multipliers and divisors. The beat histogram can be interpreted as a frequency domain representation of the novelty function.

In contrast to a magnitude spectrum computed from the audio signal itself, a beat histogram focuses only on the rhythmic content. Therefore, it is computed from a novelty function with a time window considerably longer than a typical audio transform. While this representation does not visualize the order of events in time as it discards phase information, it is a good visualization of the rhythmic periodicities in the audio signal. Each peak indicates a periodic component of the signal. The higher the peaks are, the stronger this periodic component is in the signal. The strongest peaks have been suggested to indicate the dominant tempo of the audio signal [57].

---

[3] MIREX 2018. https://nema.lis.illinois.edu/nema_out/mirex2018/results/aod/resultsperclass.html. Last retrieved on 10 August 2021.

There are multiple ways of computing such a beat histogram.

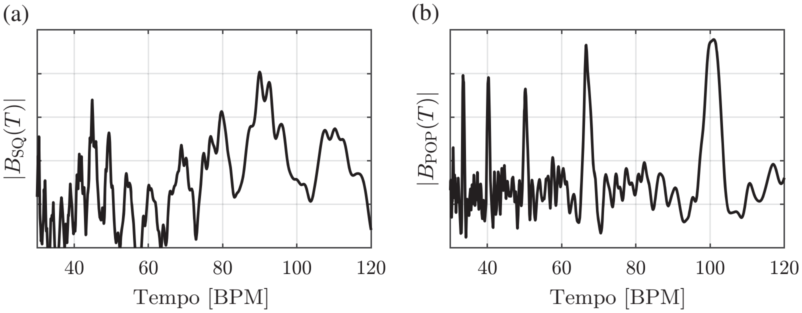
Scheirer applied a closely spaced filterbank of comb resonance filters to multiple novelty functions and used the filters' output energy as indication of the *beat strength* [38]. A similar system based on comb resonance filters has also been used by Klapuri [58].

Foote and Uchihashi proposed to construct a similarity matrix (compare Section 9.7.1) from the cosine distance between all pairs of STFTs from the audio file and then derive the beat histogram by summing the similarity matrix parallel to its diagonal at different lags [59].

Tzanetakis and Cook split the audio signal into four octave bands and extract the envelope per band by applying four processing steps [60], namely Full-Wave Rectification (FWR) by computing the absolute value, envelope smoothing by low-pass filtering, down-sampling to reduce the complexity, and DC removal by subtracting the arithmetic mean. An autocorrelation function (ACF) (with special harmonic processing) is then computed in order to identify (rhythmic) envelope regularities. Their beat histogram is constructed by taking three peaks in the search range and adding their amplitude to the beat histogram, a process applied to each texture window.

These examples indicate that a beat histogram can be computed from various novelty functions and frequency transforms.

Figure 9.7 visualizes the beat spectrum of an excerpt of popular music (b) in comparison with the spectrum extracted from a string quartet performance (a). Note that for this plot, the beat spectrum calculation is based on a very simple novelty function derived in the time domain from the signal's magnitude. After computing the ACF, the amplitude of each individual lag is mapped into the beat



Repo 1: plotBeatHistogram.m

Figure 9.7 Beat histogram of a string quartet performance (a) and of a piece of popular music (b).

domain as the beat strength (this leads to a high resolution at low frequencies and a low resolution at high frequencies). The beat histogram computed from the popular music example has clearly defined peaks at integer ratios of each other; such a pattern is not identifiable in the beat histogram computed from the string quartet recording.

9.4.1 Beat Histogram Features

Similar to an audio magnitude spectrum, various features can be extracted from the beat histogram. Examples of such features have been introduced by Tzanetakis and Cook [60]:

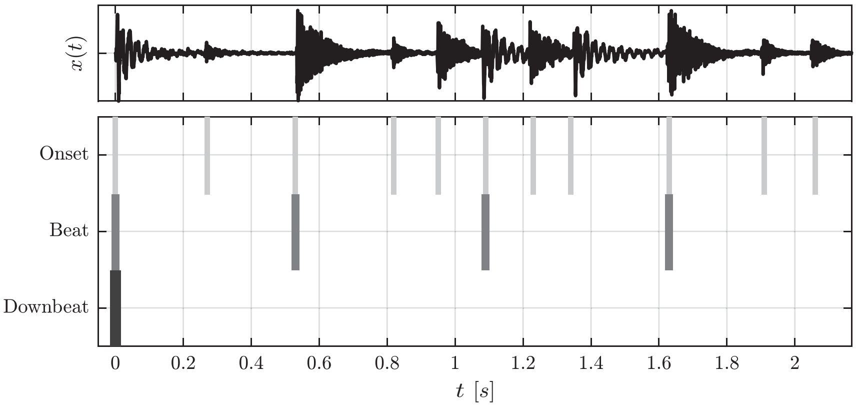
- the overall sum of the histogram,
- the relative amplitude of the highest peak,
- the relative amplitude of the second highest peak,
- the amplitude ratio of second highest to highest peak, and
- the BPM frequencies of the highest and second highest peak.

Burred and Lerch evaluated a feature set including statistical features of the beat histogram such as its arithmetic mean, standard deviation, kurtosis, skewness, and entropy; they additionally used a measure for what they called *rhythmic regularity*. The rhythmic regularity is a measure of how much the ACF of the novelty function differs from the linear weighting function of a blockwise ACF [61]. A study of all features above plus additional low-level features found it difficult to identify single powerful beat histogram features; still, the combination of features resulted in higher discriminative power when using the features for music genre classification [62].

9.5 Detection of Tempo and Beat Phase

Tempo in music is created by an underlying pulse with *beats* at regular intervals as introduced in Section 9.1.2. This pulse can be visualized as a beat grid structuring the audio signal. While an onset is a single event that can be detected without musical context, the beat only exists as an event in the context of this pulse. Sometimes the terms onset and beat are confused as they both mark a point in time; it is important to note that while an onset may coincide with a beat, there can be both onsets at time positions without beats and beats at positions without onsets. Figure 9.8 visualizes onset, beat, and downbeat positions for an example drum loop with length 1 bar.

Knowledge of the tempo and the beat grid enables a variety of applications. On the one hand, they can be used as a feature or side information for various ACA



Repo 1: plotBeatGrid.m

Figure 9.8 Visualization of Onset, Beat, and Downbeat times for a drum loop (top) of 4 beats length.

tasks. For example, tempo might be one indicator for music genre or emotion, providing important information for the automatic classification. The knowledge of the beat grid has also been successfully used as presegmentation info for chord recognition or structure detection. On the other hand, the information on tempo and beat grid can also be of direct use to a user. To give two examples from the Disk Jockey (DJ) world: the DJ music database is often annotated with tempo labels in order to enable quick identification of songs with compatible tempi. Musically intelligent software is then able to utilize the beat grid to synchronize and mix two or more pieces of music with different musical content (but the same tempo) at their beat positions to generate a so-called "mash-up" [63, 64]. Without the knowledge of the exact beat positions, it would not be possible to automatically mix those pieces in a musically meaningful way. The beat grid also allows to align a score representation with an audio file of the same piece of music, allowing, for example human–computer performances or the comparative analysis of music performance.

While the local tempo can be directly inferred as the distance between two neighboring beats on the beat grid (also referred to as *beat period*), the estimation of the tempo does not necessarily require the absolute grid position, only the distance between beats.

Traditional systems for *tempo detection* (also referred to as *tempo induction* or *tempo tracking*) usually compute some kind of novelty function as introduced in Section 9.3.1 in a first processing step. Under the assumption that musical events are both stronger and more frequent on-beat than off-beat, the distance between beats is detectable by extracting the periodicity of the novelty function. Note that, as mentioned above, the detection of specific beat positions is not necessarily required for detecting the tempo itself as the tempo is based on the *distance* between the beats rather than their absolute position. Typical examples for such periodicity analysis on the novelty function are (i) detecting the maximum of the beat histogram [38,57,58] or the maximum of an IOI histogram [65], (ii) detecting the position of the ACF maximum as an indicator of the inter-beat-distance [66], and (iii) finding the highest value of cross correlation functions (CCFs) between the novelty function and a set of quantized template delta pulses for various tempi [42]. It is worth pointing out the general algorithmic similarities between tempo detection and fundamental frequency detection. Both attempt to find regular, periodic patterns in the signal represented either in the time domain or in the frequency domain. The main differences can be found in the frequency range to detect (tempo: $\approx 1 - 3$ Hz, f_0: $\approx 50 - 2000$ Hz) and in the type and resolution of the signal to analyze (novelty function vs. audio).

The tempo indicates the distance of the beats while their absolute position is often referred to as the *beat phase*. Both are needed to create a beat grid. Systems estimating both characteristics simultaneously are often referred to as *beat tracking* systems (although some reserve this phrase for real-time systems only).

Early beat tracking systems focused on extracting the beats from symbolic data such as MIDI files. This allows for accurate onset times as input, avoiding any error propagation from computing the novelty function or the peak picking phase [67, 68]. The basic approaches are often similar in both MIDI and audio domains.

Many basic systems for beat tracking have in common an adaptive approach. This recurrent approach is shown in Figure 9.9. After starting with an initial assumption of tempo and beat phase, the next beat positions are predicted based

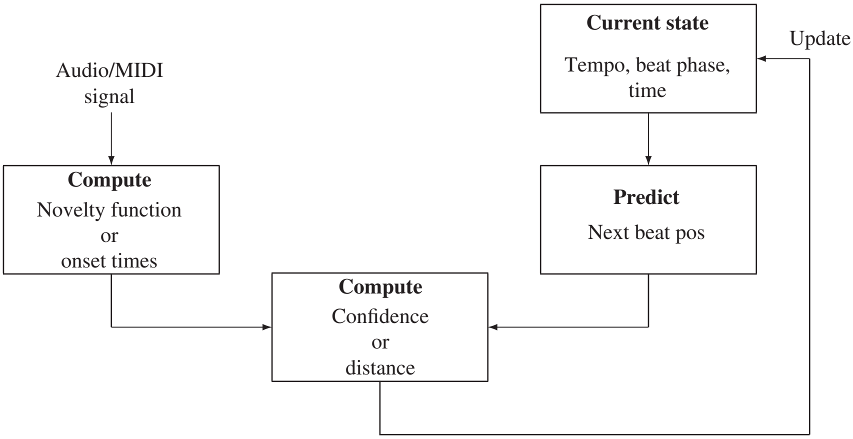


Figure 9.9 Flow chart of a typical (real-time) beat-tracking system.

on this assumption, followed by the computation of a confidence of the predicted beat location to be correct based on a comparison to either the novelty function or the series of onset times. Finally, the estimated tempo and beat phase are adapted accordingly for the prediction of the next beat. An advantage of this continuous adaptation is that the system should be able to follow moderate tempo changes over time. Large, for example, used an oscillator for generating pulses; the oscillator's beat period and beat phase are constantly adapted for a better match to the onset times [68]. The adaption speed is based on the distance between the estimated beat position and the actual onset position.

For this approach to work properly, it is crucial to have an initial estimate of tempo and beat phase that is reasonably close to the actual values. If this is not the case, a recurrent system may fail or take a long time to converge to the correct values. One way of addressing this issue is to run multiple agents with different parametrizations in parallel [45, 69]. Each agent has its own hypothesis of the tempo and the beat phase and computes its own reliability by measuring the coincidence of the estimated beat positions with the (extracted) onset positions. The agent with the highest reliability is chosen as the one providing the most likely tempo and beat phase estimate. This approach can also be extended by detecting changes in the tonal components to get additional information on the salience of onsets and estimated beats [70, 71]. Similarly, Dixon chooses between multiple agents by measuring the regularity of the IBIs and the salience of the chosen onsets [72].

In addition to onset times, beat tracking systems have also been shown to benefit from additional side information such as intensity and pitch [73, 74].

While most of the approaches presented above are (at least theoretically) capable of real-time processing, other approaches attempt to improve the robustness of tempo and beat detection by applying path searching or Dynamic Programming (DP) methods.[4] For example Allen and Dannenberg propose a beam search to select the most likely beat phase hypothesis [67]. Laroche used the most salient 10–15 maxima of the CCF between the novelty function and a template set of quantized delta pulses as initial tempo candidates. He then applies *DP* techniques to find the most likely overall path through all the tempo candidates over the whole audio file [42]. A similar approach has been introduced by Peeters [75]. He adds three different meter templates as possible states to the tempo candidates and finds the most likely tempo path through the audio file with the *Viterbi algorithm*. The usage of noncausal DP algorithms has the advantage that by finding the optimal global solution throughout the whole piece, sudden changes in local tempo can be detected [76].

---

[4] Compare the Viterbi algorithm in Sections 7.6.2 and DTW in Section 10.1 for examples of DP approaches.

Most modern, DNN-based approaches are based on recurrent network architectures. There are indications that targeting the prediction of more than one tempo-related parameter improves network performance, such as estimating beats and downbeats jointly [77] or predicting both beats and tempo in a multitask structure [78].

9.5.1 Evaluation

Given usually one ground truth-value per file, the evaluation of a tempo detection systems is straightforward as long as the tempo is assumed to be constant. Some researchers make the case, however, that *perceptual* tempo – as opposed to tempo in music which is defined by the score – is not one value but a distribution with sometimes two salient maxima [79]. This would mean that an algorithm might or has to produce two tempo results which have to be evaluated either in a separate or in a weighted combination.

The evaluation of beat tracking and the resulting beat grid is somewhat less clear, and this is reflected by the different custom metrics that have been proposed over time. The ambiguity introduced by the second perceptual tempo mentioned above poses additional questions about what an evaluation should output if, for example, all beats are correct but at half the ground truth rate.

9.5.1.1 Metrics

To evaluate the tempo detection, simple classification evaluation metrics such as the accuracy or the F-measure (see Section 6.1.1) can be computed by counting the correct and incorrect estimates given a tolerance around the ground truth (examples for this tolerance are, e.g. 4% deviation or ±1 BPM). The evaluation might also take into account that octave errors are graceful errors and calculate secondary accuracy metrics counting double, triple, half, and third tempi as correct as well [80].

The F-measure is also a useful standard metric for evaluating the beat grid, assuming a tolerance window around each beat. This tolerance window can be a fixed time (e.g. Holzapfel suggests ±70 ms [81]) or tempo-dependent (e.g. Dixon suggests a fraction of the IBI [82]). The following other metrics have been proposed:

- A weighted distance $\Delta t_{\mathrm{R,D}}$ between reference and detected time with a Gaussian window function $w(\Delta t)$ [83]:

$$q_{\mathrm{cemgil},2} = \frac{2\sum_{\forall r} \max_{\forall t} w(\Delta t_{\mathrm{R,D}})}{TP + FN + TP + FP}. \tag{9.11}$$

- The P-score, computing a binary match over the quantized time, normalized by the maximum of the number of detected and the number of ground truth beats, and averaging the results over multiple annotators [84].

- The *Information Gain* of a normalized beat error histogram over a uniform distribution, computed as the Kullback–Leibler Divergence between these two distributions [81].

Davies and Böck compare these and more metrics with subjective ratings [85]. While the results are largely inconclusive, the authors warn against only using the F-measure or the P-score and encourage the use of additional metrics such as the information gain.

9.5.1.2 Datasets

While a beat detection dataset can also serve as data for the evaluation of tempo detection, this is not true the other way around. Most datasets focusing on tempo-only assume constant tempo and provide only one tempo value per song (snippet). Examples for such datasets are Extended Ballroom [86] consisting of dance music, GiantSteps [87] focusing on electronic dance music, and GTZAN-Tempo [88], containing song snippets from a variety of genres. Beat Annotations are available for, e.g. the Ballroom [80] and SMC [81] datasets.

9.5.1.3 Results

Due to the hierarchical nature of rhythmic and structural properties of music, the most common errors of beat tracking systems are so-called “octave errors,” i.e. detecting the half or the double tempo. Obviously, this also depends on the tempo range that the system is configured to detect; the smaller the target range, the less likely are octave errors.

The variety of testing data leads to large variations of results for different datasets. Tempo detection accuracy of state of the art systems tends to vary roughly between 60% and 90%, but tends to be higher than 90% if octave errors are ignored [78].

The biggest challenges to most tempo tracking systems are varying tempi and input signals without strong and easily identifiable onsets. In these cases, F-Measures around or even below 50% can be considered good, compare the Audio Beat Tracking results of MIREX 2019.[5]

9.6 Detection of Meter and Downbeat

The relation of *meter* and *downbeat* is very similar to the relation of tempo and beat phase. Just as the tempo is derived from the distance between two neighboring beats, the meter can be derived from the distance between two downbeats

---

[5] MIREX 2019. https://www.music-ir.org/mirex/wiki/2019:MIREX2019_Results. Last retrieved on 4 July 2021.

(length of a bar), while the downbeat position marks the start time of a bar. Detecting the length of a musical bar also helps with detecting the time signature when combined with beat information.

Downbeat and meter information can be helpful for a variety of tasks. For example, the knowledge of downbeat positions can help to improve beat matching different songs. It also helps as a preprocessing step for tasks such as structure detection since a bar is a fundamental building block of musical structure.

The hierarchical metrical structure of music makes the differentiation between beat detection and downbeat detection basically a question of the hierarchical level to investigate: meter and downbeat detection requires a refocus on long-term periodicities, but the approaches are similar with differences in the computation of the novelty function and the search range for periodicities.

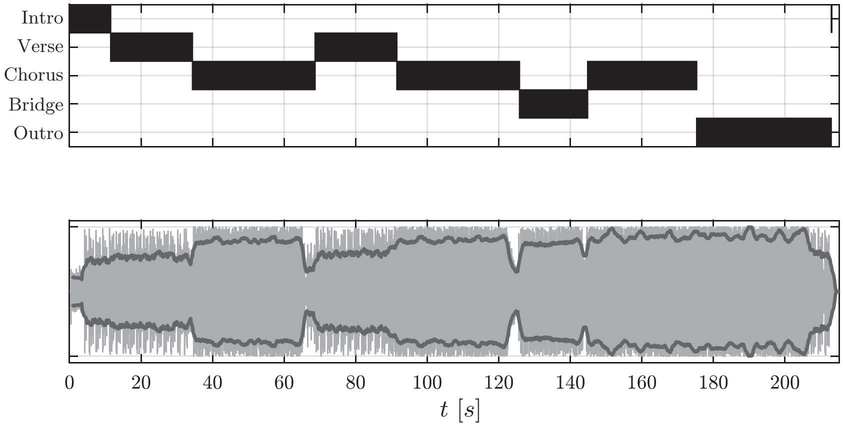
Brown weighted the series of onsets with their IOI (in order to increase the impact of long notes) and computed the ACF of this series of weighted onsets to detect the meter [89]. Toiviainen and Eerola proposed a similar approach and evaluated different weighting functions for the IOIs [90].

Uhle and Herre derived bar length candidates from integer multiples of a previously detected tatum and then computed the CCF of two snippets of the novelty function per frequency band to derive a measure of the likelihood of the individual bar length candidates [19]. Using the tatum has the two advantages of a signal-adaptive segmentation and higher computational efficiency of the following processing steps.

Escalona-Espinosa argued that it is not only the onset pattern itself that is of interest for the estimation of meter and downbeat, but that other features should be used for computing the novelty function as well [91]. More specifically, he assumed that in the Western tradition of music (and even more so in the case of popular music), the position of a downbeat increases the likelihood of both (i) a note or harmony change and (ii) the occurrence of a new bass note compared to positions between downbeats. Based on this reasoning, he proposed the computation of two novelty functions, one based on the pitch chroma difference and the second on the bass energy increase. The time resolution for this computation is set to a previously estimated tatum. As an alternative to ACF-based approaches, he constructed two Self-Similarity Matrices (see Section 9.7.1) from the two novelty functions. A Self Similarity Matrix (SSM) contains the similarity between all pairs of samples in a sequence with the diagonal having the maximum similarity. When averaging the parallels of the diagonal of each similarity matrix, the result is a measure of periodicity with respect to the distance from the main diagonal. Depending on the similarity (or distance) measure used for the computation of the SSM, this function has similar meaning as the ACF, as the lag of the main peak within a predefined search range is the estimated bar length. In combination with a tempo estimate, the result thus allows us to derive the time signature of the piece of music. The most likely downbeat position is then estimated with the extracted bar length by computing the CCF of each novelty function with a delta pulse spaced with the bar length period. The lag of the CCF's maximum indicates the downbeat position.

9.7 Structure Detection

Similar to how a book has form on different levels (words are grouped into sentences, sentences are grouped into paragraphs, paragraphs are grouped into sections, and sections are grouped into chapters), music has hierarchical structural groupings; it has been argued that form in music even plays a more pronounced role than in literature [92]. We have already looked into different hierarchical levels of grouping above where tatum, tactus, and meter have been introduced. But there are many more large-scale structural elements in music. In Western classical music, bars are grouped into phrases and phrases are grouped into sentences or periods [93].[6] The highest grouping levels are usually those of a movement and, ultimately, the complete piece, for example, the symphony or sonata. Western popular music has similar fundamental building blocks (phrases) and combines them into segments such as chorus and verse. Figure 9.10 shows an example structure of a popular song. Studying the structure and form of music is a common task in Western music theory, but while the general structural elements are clear, the actual result of a structural analysis is subjective even



Repo 1: plotStructure.m

Figure 9.10 Example structure of a popular song (here: Pink's *So What*).

---

[6] Note that there are alternative terminologies commonly used.

if experts often consent. The potential ambiguity of musical structure stems from the fact that many different factors inform the listener of structure and structural boundaries. There is the perceptual grouping of note events based on rhythmic and melodic patterns, a harmony progression with cadences potentially marking structural boundaries, as well changes in key, tempo, or instrumentation marking new segments. Higher-level characteristics like tension and relaxation, which combine rhythmic, harmonic, dynamic, melodic, and timbral elements can also impact the perception of musical structure [94]. Finally, there is also the performers' rendition of the piece which can reinforce or even shape the perception of structure in music through accents and tempo variations [95]. Structure can be inferred from the audio signal by looking at three main characteristics [96], (i) novelty and contrast (meaning something new and possibly unexpected happens), (ii) homogeneity (meaning that similar parts tend to be grouped together), and (iii) repetition (meaning that the recognition of a repeated segment indicates a structural segment). All of these characteristics can be represented through a variety of musical elements including but not limited to harmony, rhythm, melody, instrumentation, tempo, and dynamics. A structural boundary could, for example be indicated by an instantaneous change of rhythmic patterns, dynamics, or instrumentation (high novelty). A segment is often more homogeneous within itself with respect to tonality, harmony, or tempo than in contrast to other segments. The repetition of a sequence of bars marks structural boundaries and indicates a potentially important segment.

Possible applications of structure detection can be found in visualization and navigation within a song; for example a DJ may want to jump to specific segments in a song during a performance. Structural information can also be used for large-scale musicological analysis and might support other ACA tasks such as downbeat detection or audio classification.

The objective of automatic structure detection is to reveal structural properties and relationships in a way that a list of musically meaningful, nonoverlapping segment boundaries can be extracted. This list should also indicate segment repetitions and musical relations between segments. Note that the goal is not necessarily to assign labels such as "verse" or "chorus" to the detected parts, although some systems attempt that. An overview article on structure analysis was published by Nieto et al. [97]. Music analysis tasks related to structure detection are chorus detection [98] and audio thumbnail extraction [99].

9.7.1 Self-Similarity Matrix

A common input representation of a structure detection system is the *SSM*. The SSM visualizes the similarity between different instants of a piece of music in an intuitive way. Given a feature sequence $v(n)$, the elements of similarity matrix $\boldsymbol{S}$ can be computed by

$$S(n_\mathrm{A}, n_\mathrm{B}) = \mathrm{s}\left(v(n_\mathrm{A}), v(n_\mathrm{B})\right) \tag{9.12}$$

with s being a measure of similarity, e.g. the cosine similarity.

Both axes of the resulting SSM indicate time. The SSM is symmetric, as the similarity between $(n_\mathrm{A}, n_\mathrm{B})$ is identical to the similarity between $(n_\mathrm{B}, n_\mathrm{A})$ (for symmetric similarity measures). Figure 9.11 shows the SSM for a popular song as an example. Note that while structural details can be seen for this real-world recording, analyzing it with the simple approaches presented in this chapter will only have limited success. The bright diagonal indicates the maximum of self-similarity $(n_\mathrm{A} = n_\mathrm{B})$. Areas of constant bright color indicate areas of high homogeneity such as held chords or short repeated patterns, and dark vertical or horizontal lines indicate low similarity to all other points (often breaks and pauses). Lines parallel to the diagonal indicate repetitions with the distance to the diagonal being the time between repetitions. Analyzing the specific SSM in Figure 9.11 more closely, the following structural indicators can be identified visually: the intro stops at about 19s, the bridge at 60–69s is followed by a chorus (69–86s), and the same constellation is repeated (as indicated by the line parallel to the diagonal) at 120s, the high homogeneity of the instrumental is indicated by the bright box from 145 to 170s, and the song ends with four repetitions of the chorus starting at 180s visualized

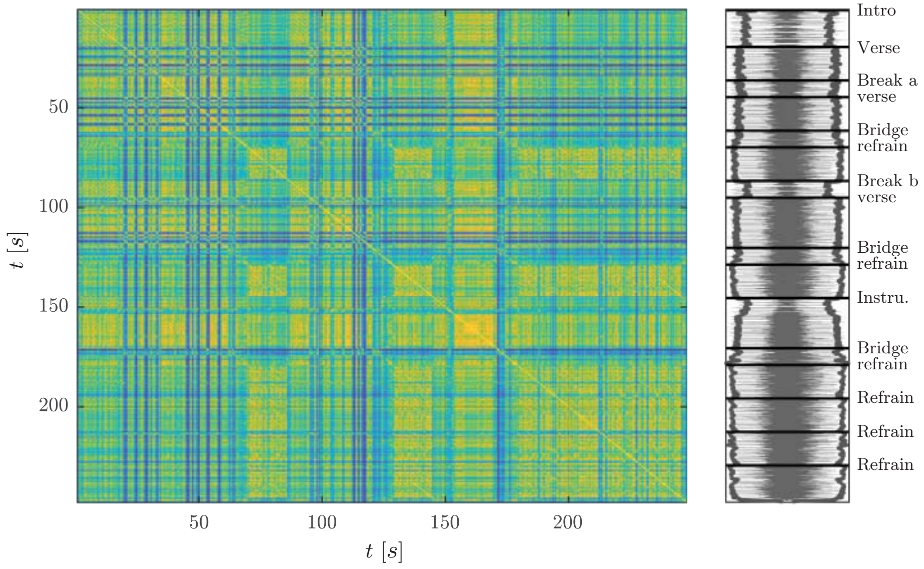


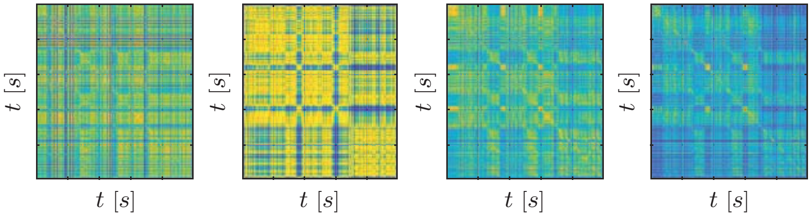
Figure 9.11 Self-Similarity Matrix of Michael Jackson's *Bad*.

by the high similarity with diagonal structure in the end. Note that the above-mentioned lines parallel to the diagonal cannot be observed for recordings with varying tempo. While repetitions which replicate tempo exactly will result in such parallel lines, repetitions with different tempo will "bend" such lines slightly up or down.

The choice of the input feature sequence depends on the musical characteristic targeted for similarity. The example in Figure 9.11 has been computed with the pitch chroma – a 12-dimensional feature representing the octave-independent pitch content (see Section 7.5.1) – but the self-similarity can be computed with any meaningful musical feature describing pitch and tonal content, rhythmic content, timbre content, or dynamics. Figure 9.12 displays the SSMs for the same piece but computed from different input representations, namely (from left to right) pitch chroma, RMS, MFCCs, and the Mel Spectrogram. Overall, these matrices look similar when we discard the skewed distributions of the similarities as indicated by the different average brightness. A closer look, however, reveals that some structural elements are more or less clearly highlighted by specific features. Ultimately, this is not surprising as, for example, verse and chorus might have the same level but different tonal content. It is a good reminder that there is no general objective measure of overall music similarity; rather, the similarity has to be computed for multiple specific, clearly defined characteristics.

The parametrization, especially the time resolution of the SSM can impact the usefulness of the matrix considerably. It is possible to compute multiple matrices with different time resolutions to find a balance between accurate timing and good overall detection; however, the timing accuracy of boundary detection might also be improved by utilizing other ACA tasks such as beat detection and aggregating the feature values over the beats.

The raw SSM is often subjected to postprocessing. Common examples are (i) filtering, such as the application of a low-pass smoothing filter or a high-pass edge detection filter, (ii) applying a nonlinear function to the matrix values, e.g. to decrease lower values or increase higher values, or (iii) quantizing the



Repo 1: plotSsmFeatures.m

Figure 9.12 Self-Similarity Matrices of the same song based on different features (from left to right: pitch chroma, RMS, MFCCs, and Mel Spectrogram).

matrix values to a binary matrix by thresholding, thus enabling subsequent image processing techniques such as erosion and dilation [100].

9.7.2 Approaches to Structure Detection

There are several ways to use the SSM to algorithmically infer the musical structure. As pointed out above, the main approaches revolve around the detection of novelty, homogeneity, and repetition.

9.7.2.1 Novelty Analysis

The principles behind the extraction of novelty function from music signals have been introduced in the context of onset detection above (see Section 9.3). The differences to this structural analysis are that, on the one hand, significantly longer time windows have to be considered and that, on the other hand, many other low- or high-level features can be helpful for extracting the novelty function.

A simple way of extracting a novelty function from the SSM is sliding a checker board kernel as visualized in Figure 9.13 along the matrix diagonal [32]. The checker board kernel has a high-pass filter characteristic. Sweeping it along the diagonal, the filter output should be high in the case of a change in the SSM and low without boundaries. The resulting novelty function is displayed in Figure 9.14.

9.7.2.2 Homogeneity Analysis

Detecting homogeneous regions in the SSM has led to a multitude of different proposals in the past. A naive approach might try to identify regions of negligible change by either segmenting the novelty function according to regions close to zero or detect plateaus in a low-pass filtered diagonal. The result of such a low-pass filter is shown in Figure 9.15. As can be seen, this naive approach does not give convincing results with the chosen features and parameters for this example signal.

Other approaches have successfully used clustering methods for both detecting neighboring SSM entries [101] as well as identifying other segments that are repetitions [102].

9.7.2.3 Repetition Analysis

Repetitions, indicated by lines parallel to the diagonal, might be found with edge detection techniques on the SSM [103]. A common approach to simplify the edge

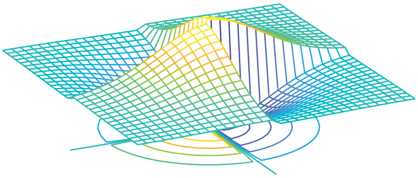
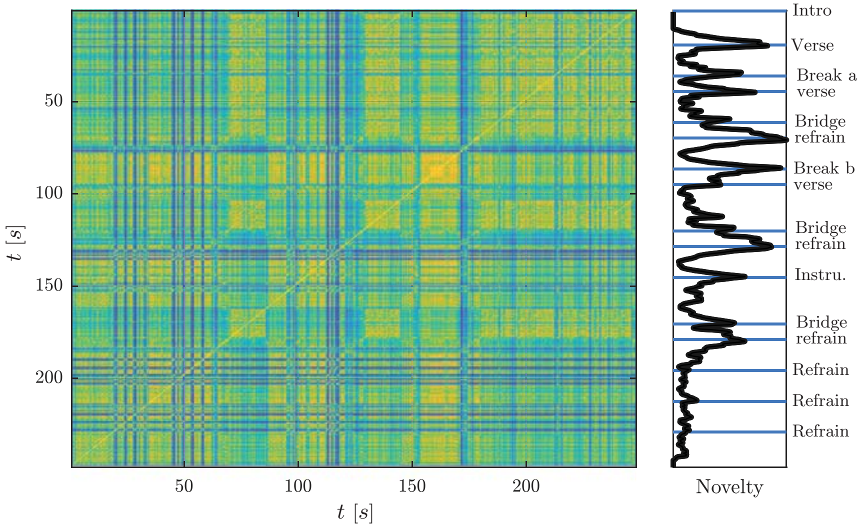


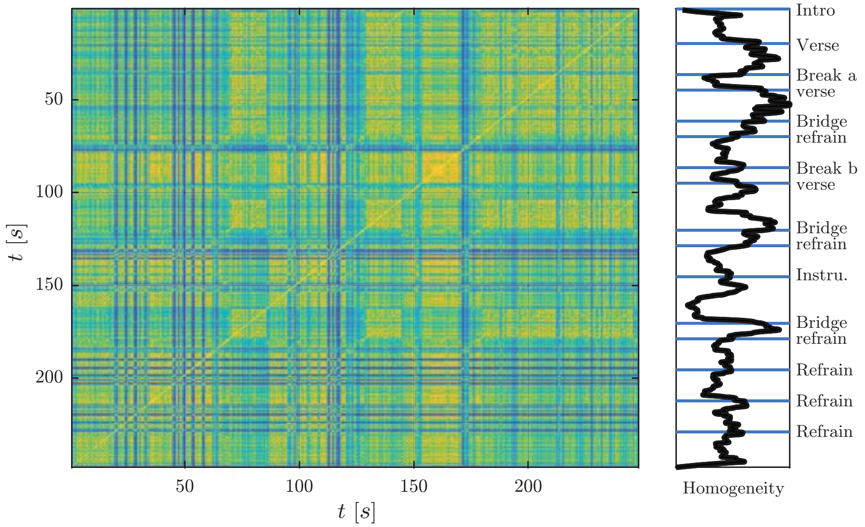
Figure 9.13 Checker Board Filter Kernel with high-pass characteristics to detect novelty in a SSM.

Repo 1: plotCheckerBoard.m



Repo 1: plotSsmNovelty.m

Figure 9.14 Self-Similarity Matrix (left) and extracted Novelty Function.



Repo 1: plotSsmLowPass.m

Figure 9.15 Self-Similarity Matrix (left) and Low-pass filtered Diagonal as an estimate of the Homogeneity Function.

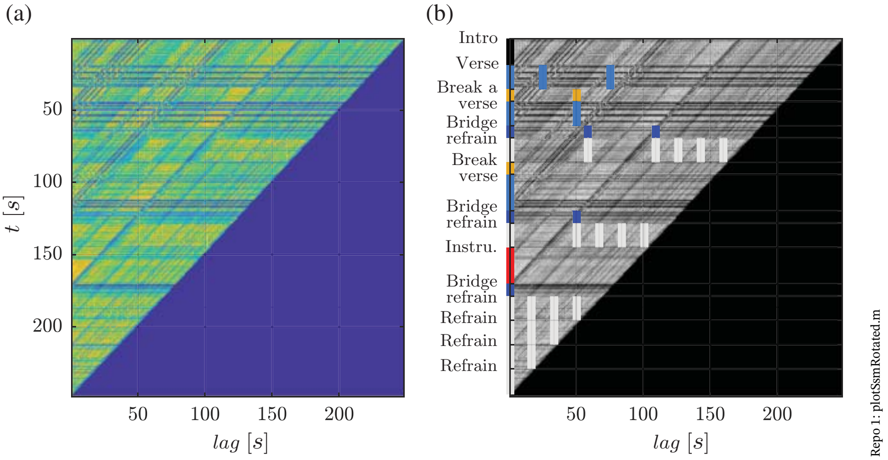


Figure 9.16 Rotated Self Similarity Matrix (a) and accompanying Ground truth (b) for detection of repetitions.

detection is to rotate half of the SSM in a way that one axis shows the distance from the diagonal as lag. All parallels to the diagonal of the original SSM will then be shown as vertical lines. Figure 9.16 shows such a rotated matrix (a) and the accompanying ground truth labels (b). Note that the rotated matrix in this example has not been extensively postprocessed so the structure is not unambiguously identifiable in this example. The ground truth labels use different colors for different parts; parsing the lag 0 from top to bottom directly visualizes the ground truth song structure. Parts which are not repeated (such as the intro or the instrumental) lack any recognizable vertical lines at lags larger than 0. Parts that are repeated, however, show vertical lines at the lag corresponding to the time between the repetitions. Note that there is a certain redundancy to this visualization: for example the four repetitions of the refrain at the end of the song appear as repetitions of the first refrain (at ≈ 70 seconds) with a lag of 120 seconds but also as repetitions of the second refrain (at ≈ 130 seconds) with a lag of 60 seconds. These patterns can be utilized in the design of an algorithm detecting repetitions as similar patterns of vertical lines will appear multiple times.

9.7.3 Evaluation

The formal evaluation of structure detection systems is more complicated than it seems at first glance, as human annotators – even if in agreement about the structure – tend to annotate different structural levels. Table 9.1 visualizes three hypothetical annotations by different annotators for the same piece of music. All three annotations can be correct, however, simply matching one of the ground truths to the detected structure does not take into account that the system might have

Table 9.1 Hypothetical example for annotator disagreements on musical structure.

Intro	A				A				Outro
Intro	Verse		Chorus		Verse		Chorus		Outro
Intro	V_1	V_2	C_1	C_2	V_1	V_2	C_1	C_2	outro

detected the structure correctly at a different level. Despite repeated proposals of addressing this problem [104–106], there remain open questions how to best deal with this hierarchy issue [97].

To complicate matters more, the structure itself may be ambiguous. As structural boundaries can be conveyed by a multitude of different indicators, listeners might not be in complete agreement as where structural boundaries are. Annotations are also complicated by a simpler form of subjectivity involved, for example the definition of musical repetition: if the repetition of a chorus is modified in some way, is it still labeled as the chorus or not?

9.7.3.1 Metrics

The ground-truth complications with ambiguous, nonmatching labels, confusion about hierarchical level, and annotator disagreement impact the choices of metrics for structure detection.

To evaluate the predicted boundaries, most commonly the F-measure, see Eq. (6.5), is used. A predicted boundary is a TP if a ground truth boundary is within a certain tolerance (usually 0.5–3 seconds), a FP otherwise. Confusingly, this F-measure is occasionally referred to as the *Hit Rate* [107].

If blockwise processing allows each block to be assigned to a specific segment, then the validity of these assigned labels can be evaluated with *Pairwise Clustering*. Again, this is an F-measure, with the difference that a TP is now found when a pair of blocks with the identical estimated labels is verified to have identical labels in the ground truth as well [101].

In addition, metrics based on conditional entropies have been proposed to measure over- and undersegmentation [108].

9.7.3.2 Datasets

While there is a number of datasets available for structure detection, the availability of audio files is – similar to other tasks – often limited. The only datasets with publicly available audio data with segment annotations are the RWC dataset [109] and a small dataset called Sargon.[7]

---

[7] Oriol Nieto. https://github.com/urinieto/msaf-data/tree/master/Sargon/audio. Last retrieved on 12 August 2021.

For the remaining datasets, audio data have to be acquired in different ways. This can result in other complications as the same songs may be published in different versions and masters, leading to time shifts and potentially more severe discrepancies between audio and annotations. Thus, all annotation data has to be reverified by the researchers for each audio file. Frequently used datasets for segmentation include the Harmonix Set, the INRIA data [110], Isophonics [111], SPAM [112], and SALAMI [113].

9.7.3.3 Results

Recent results are not easily available or comparable. Results from previous MIREX years indicate that the pairwise F-Measure will – dependent on the evaluation data – be roughly in the range of $50 - 70\%$ with large error margins.[8]

9.8 Automatic Drum Transcription

Automatic Drum Transcription (ADT) is a task that can be interpreted as either a classification or transcription problem. On the one hand, it targets classifying different drum instruments being active, on the other hand, its goal is to transcribe the exact times percussion instruments are being hit. It is probably most intuitive to describe the task drum transcription to be a special case of onset detection, where only percussive onsets are detected, and each onset is assigned to one or more percussive instruments. An exhaustive survey of ADT approaches can be found in Wu et al. [114].

The applications of ADT range from (i) improving other ACA tasks targeting higher-level musical data based on drum patterns such as rhythm detection, the estimation of danceability, or the separation of audio sources in a mix, and (ii) supporting educational software by providing detailed instrument and timing information for music tutoring systems, to (iii) enabling creative use cases, for example swapping drum sounds in a mixture or enabling targeted audio effect processing of specific drum hits in a mixture.

Similar to other tasks, there exist multiple subtasks with varying complexity that are commonly referred to as ADT and the easier tasks tend to be addressed earlier in the literature. The task complexity is mostly determined by the complexity of the input data: the classification of individual, isolated drum sounds was of interest in many early publications on the topic [115–119]. The transcription of drum-only recordings marks the next complexity level, detecting the onset times for each drum instrument while allowing multiple synchronous drum onsets

---

[8] MIREX 2014. https://www.music-ir.org/mirex/wiki/2014:MIREX2014_Results. Last retrieved on 12 August 2021.

[54, 120–123]. Most modern systems tend to focus on the general task of transcribing onset times for each drum instrument from a recording of multiple percussive and nonpercussive instruments [124–136]. The main challenge these systems face is the detection of nonpitched drum sounds in a complex mixture of pitched and nonpitched sounds. This is further complicated by the fact that drum hits can occur simultaneously and that some percussive instruments occur much more frequently than others. For these reason, most of the literature on ADT focuses on the detection of just three drum instruments *Bass Drum*, *Snare Drum*, and *Hi-Hat*, although attempts have been made toward a more exhaustive set of percussive instruments [137, 138]. It should be noted that research on ADT almost exclusively focuses on drums as used in Western popular music – other percussive instruments and nonpopular genres are mostly ignored by the literature.

9.8.1 Transcription of Drum Onsets

Simple baseline approaches revolve around modifying an onset detection system (as introduced in Section 9.3) so that it can also classify drums. This is done by either applying a preprocessing step or a postprocessing step to the onset detection system; Figure 9.17 shows a visualization of these two basic modifications. A simple preprocessing step visualized at the top is to apply a filterbank with three bands to the audio signal before extracting three novelty functions (in drum transcription also referred to as the *activation function*. As each filter band targets one drum instrument, each novelty function represents the activation of this instrument. A standard peak picking approach can be applied to detect the onsets per band and thus transcribe the drum. The bottom of Figure 9.17 shows that the output of a standard onset detection algorithm can be used to segment the audio file and classify each segment into a drum instrument in a postprocessing step. This approach is complicated by the fact that drum transcription is a multilabel

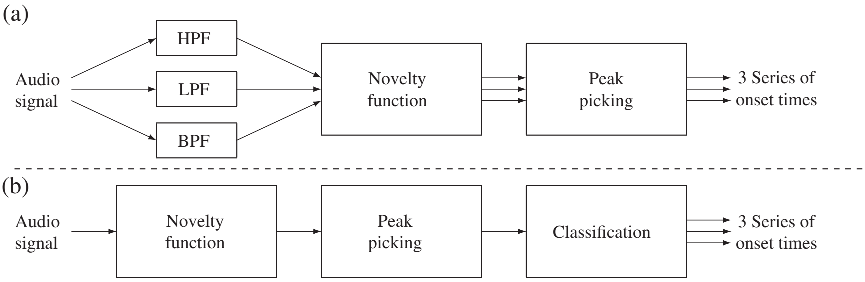


Figure 9.17 Two baseline drum transcription systems by extending a standard onset detection approach: (a) applying a filterbank to the input signal to extract three novelty functions, (b) adding a classification postprocessing step after the peak picking.

problem, i.e. all classes can be activated simultaneously, as opposed to a standard single-label classification, in which only one class can be active for each observation. Therefore, either a single-label classifier is used with not only three single classes, but all possible combinations of simultaneous drums, or a multilabel approach has to be applied. Both baseline approaches are not particularly well suited to deal with complex mixtures with nondrum sounds. Other approaches that have been shown to be more robust in that scenario are Non-negative Matrix Factorization (NMF)-based approaches (compare Section 7.3.4.2). NMF is an iterative algorithm factorizing the input spectrogram into template components and their corresponding activation functions; after the templates for each drum instrument are identified, peak picking can be applied to the corresponding activation functions. It is also possible to keep three drum templates fixed and only apply the factorization operation to the rest of the templates and the activations [133].

One of the biggest current challenges in ADT is data. As modern, neural network-based models need large amounts of accurately annotated data, but the generation of new expert-annotated datasets requires huge efforts, a considerable number of approaches try to find workarounds to this problem. These workarounds include, for instance, training with augmented data [139], applying unsupervised and semisupervised training schemes [140–142], and using generated training labels through aligned audio and MIDI files [143].

9.8.2 Evaluation

The evaluation of drum transcription systems is similar to the evaluation of onset detection systems (see Section 9.3.3), except that it is done for multiple classes.

The most commonly used datasets are the IDMT-SMT-Drums dataset [54] for drums-only classification and the ENST-Drums dataset [52] for the evaluation of mixtures. Other useful datasets include the RWC [109] and the MDB Drums datasets [55]. There have been other recent efforts to make additional datasets for the task available [144, 145].

9.9 Exercises

9.9.1 Questions

1. Sort the following instruments according to their expected attack time: flute, harpsichord, trumpet.
2. What is the relation of tempo and the distance between two neighboring beat markers?

3. Assuming a constant tempo: if a piece of music has 20 beats in 10 seconds, what is the tempo in BPM?
4. Given a time signature of 5/8, what is the number of sixteenth notes you could fit into one measure?
5. You have a rock song and a string quartet recording. Which of the signals do you expect to lead to stronger peaks in the beat histogram? Explain why.
6. A beat spectrum shows a maximum at 3 Hz. What is the most likely tempo in BPM?
7. Describe the flowchart and processing steps of a prototypical onset detection system.
8. Consider an IOI histogram with a maximum at 400 ms. What is the most likely tempo in BPM?

9.9.2 Assignments

1. Onset detection
   (a) Generate a sinusoidal test signal of length 4 seconds at a sample rate f_S = 44.1 kHz with
      - 0–1 seconds: $f = 441$ Hz,
      - 1–2 seconds: $f = 882$ Hz,

   (b) Implement a function `[novelty_flux] = extract_noveltyflux (x_B)` that is similar to the function `[v_sf] = extract_spectral_ flux(x_B)` implemented in Section 3.8.2, however, has the following modification of the equation

$$v_{\mathrm{SF}}(n) = \frac{\sqrt{\sum\limits_{k=0}^{\mathcal{K}/2} \mathrm{HWR}(|X(k,n)| - |X(k,n-1)|)^2}}{\mathcal{K}/2 + 1}$$

   to ensure energy decreases to not count.

   (c) Implement a function `[novelty_peak] = extract_noveltypeak (x_B)` that internally calls the function `[v_peak] = extract_peak (x_B, fs)` implemented in Section 8.4, then computes a simple difference on it (derivative with respect to time). If you haven't implemented it yet, use the following instructions:
      - Implement a function `[v_peak] = extract_peak(x_B, fs)` which returns the peak envelope of the blocked signal. Return the absolute maximum per block in dBFS.

   (d) Plot the two novelty functions for the input signal given above. Locate the likely onset point and discuss the suitability of each novelty function for onset detection.

- (e) Implement a function `[threshold] = compute_thresh(novelty, order, lambda)` which implements a median filter of order `order` and adds an offset `lambda` to the filter output. Be sure that your output has no delay or latency compared to the input (zero-phase). Discuss how increasing lambda or changing the order will impact the threshold.
- (f) Implement a function `[t_onset] = find_onsets(novelty, thresh)` that detects local maxima in the novelty function above the threshold and returns their location in seconds in vector `t_onset`.
- (g) Implement a function `[precision, recall, fmeasure] = eval_onsetdetection(t_onset_est, t_onset_gt, tolerance:InS)` that takes two lists of onset times (detected and ground truth) of usually different length and computes the number of false positives (falsely detected onset) and FNs (missed onset) with a tolerance window of `tolerance:InS` to compute precision, recall, and F-measure.
- (h) Download the ODB dataset[9] and evaluate your onset detection system. In case you have not implemented file reading and blocking yet, use the following instructions:
  - Implement a function `[fs, x] = read_audio(cPath2Audio)` which reads a wave audio file with scipy.io.wavfile.read and returns the sample rate `fs` and a numpy array `x` with dimensions `numSamples`× `numChannels`. Ensure that the maximum possible value range of `x` is always [−1,1[regardless of the wordlength of the input file. Support the following wordlengths/input types: `uint8`, `int16`, `int32`, `float32`
  - Implement a function `[x_m] = downmix_audio(x)` which takes a numpy array `x` with dimensions `numSamples` × `numChannels` and returns the downmixed numpy array `x_m` with dimensions `numSamples` × 1.
  - Implement a function `[x_B, t_B] = block_audio(x_m, blockSize, hopSize, fs)` which returns a numpy array `x_B` (dimension `N` × `blockSize`) and a vector t_B (dimension `N` × 1) for blocking the input audio signal into overlapping blocks. Make sure that each sample of the audio signal is included in at least one of the `N` blocks. `t_B` will refer to the start time of each block in seconds.

2. Structure
   - (a) Implement a function `[D] = compute_dist(v_1,v_2, cDistanceMeasure)` which takes two series of feature vectors `v_1` and

---

[9] https://grfia.dlsi.ua.es/cm/worklines/pertusa/onset/ODB/. Last retrieved on 27 November 2021.

`v_2` and computes the pairwise distance between them. The output `D` is nonnegative and its dimensions correspond to the number of observations in series `1` times the number of observations in sequence `2`. `cDistanceMeasure` can be

- `'euclidean'`: Euclidean distance,
- `'manhattan'`: Manhattan distance, or
- `'cosine'`: Cosine distance.

(b) Compute a SSM of an audio file based on pitch chroma features. If you haven't implemented `extract_pitch_chroma`, use the following instructions:

- Implement a function `[w] = compute_hann(windowLength)` that computes a Hann window of length `windowLength` and returns the window in numpy vector `w`. Before you implement, investigate the difference between choosing a "symmetric" vs. a "periodic" window function and choose the "periodic" implementation.
- Implement a function `[X, fInHz, t_B] = compute_spectrogram(x_B, t_B, fs)` that computes the magnitude spectrum for each block of audio in `x_B` and returns the magnitude spectrogram `X` (dimensions `blockSize/2+1` × `N`), a frequency vector `fInHz` (dimensions `blockSize/2+1` × 1) containing the central frequency of each bin, and the previously computed time stamp vector `t_B`.
- Write a function `[v_pc] = extract_pitch_chroma(X, fs, tfInHz = 440)` which returns the pitch chroma array `v_pc` (dimensions 12 × numBlocks). `X` is the magnitude spectrogram, `fs` is the sample rate, and `tfInHz` is the tuning frequency. Compute the pitch chroma for the 3 octave range C3–B5. Make sure your implementation works with different sample rates and block sizes. You will need to adjust the semitone bands based on the input parameter `tfInHz`. Normalize each individual chroma vector to a length of 1 (L2 norm).

(c) Plot the matrices for various music files and adjust blockSize and hopSize to visually optimize for structure detection. Discuss the structural properties you can see in these matrices.

(d) Implement a function `[novelty_ssm] = extract_noveltyssm(S, kernel_length)` which extracts a novelty function from SSM `S` using a check-board kernel of lengths 2, 5, 16, 32.

(e) Plot the novelty functions for your test files in comparison.

(f) Describe how you would implement a lag distance matrix for repetition detection.

References

1 Grosvenor W Cooper and Leonard B Meyer. *The Rhythmic Structure of Music*. University of Chicago Press, 1963. ISBN 978-0-226-11522-1.

2 Matthew J Wright. *The Shape of an Instant: Measuring and Modeling Perceptual Attack Time with Probability Density Functions*. PhD Thesis, Stanford University, Stanford, 2008. 00015.

3 Edward W Large and Caroline Palmer. Perceiving Temporal Regularity in Music. *Cognitive Science*, 26(1):1–37, January 2002. ISSN 03640213. URL http://doi.wiley.com/10.1207/s15516709cog2601_1.

4 Christoph Reuter. *Der Einschwingvorgang nichtperkussiver Musikinstrumente*. Peter Lang, Frankfurt, 1995.

5 Juan Pablo Bello, Laurent Daudet, Samer Abdallah, Chris Duxbury, Mike Davies, and Mark B Sandler. A Tutorial on Onset Detection in Music Signals. *IEEE Transactions on Speech and Audio Processing*, 13(5):1035–1047, 2005. ISSN 1063-6676. doi: 10.1109/TSA.2005.851998. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1495485.

6 Bruno H Repp. Patterns of Note Onset Asynchronies in Expressive Piano Performance. *Journal of the Acoustical Society of America (JASA)*, 100(6):3917–3932, 1996.

7 John William Gordon. *Perception of Attack Transients in Musical Tones*. PhD Thesis, Stanford University, Center for Computer Research in Music and Acoustics (CCRMA), Stanford, 1984.

8 Eberhard Zwicker and Hugo Fastl. *Psychoacoustics. Facts and Models*. Springer, 2nd edition, 1999. ISBN 978-3-540-23159-2.

9 Ira J Hirsh. Auditory Perception of Temporal Order. *Journal of the Acoustical Society of America (JASA)*, 31(6):759, 1959. ISSN 00014966. doi: 10.1121/1.1907782. URL http://link.aip.org/link/JASMAN/v31/i6/p759/s1&Agg=doi.

10 Anders Friberg and Johan Sundberg. Perception of Just Noticeable Time Displacement of a Tone Presented in a Metrical Sequence at Different Tempos. *STL-QPSR*, 33(4):97–108, 1992.

11 Bruno H Repp. Diversity and Commonality in Music Performance: An Analysis of Timing Microstructure in Schumann's 'Träumerei'. *Journal of the Acoustical Society of America (JASA)*, 92(5):2546–2568, 1992.

12 Pierre Leveau, Laurent Daudet, and Gaël Richard. Methodology and Tools for the Evaluation of Automatic Onset Detection Algorithms in Music. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Barcelona, 2004.

13 Rudolf A Rasch. Synchronization in Performed Ensemble Music. *Acustica*, 43:121–131, 1979.

14 L Henry Shaffer. Timing in Solo and Duet Piano Performances. *The Quarterly Journal of Experimental Psychology*, 36A:577–595, 1984.

15 Daniel J Levitin and Perry R Cook. Memory for Musical Tempo: Additional Evidence That Auditory Memory is Absolute. *Perception & Psychophysics*, 58(6):927–935, January 1996. ISSN 1532-5962. doi: 10.3758/BF03205494. URL https://doi.org/10.3758/BF03205494.

16 J Devin McAuley. Tempo and Rhythm. In Mari Riess Jones, Richard R Fay, and Arthur N Popper, editors, *Music Perception*, Springer Handbook of Auditory Research, pages 165–199. Springer, New York, NY, 2010. ISBN 978-1-4419-6114-3. URL https://doi.org/10.1007/978-1-4419-6114-3_6.

17 Peter Desain and Henkjan Honing. Tempo Curves Considered Harmful. *Time in Contemporary Musical Thought, Contemporary Music Review*, 7(2):123–138, 1993. doi: 10.1080/07494469300640081.

18 Fred Lerdahl and Ray Jackendorf. *A Generative Theory of Tonal Music*. MIT Press, Cambridge, 1983. ISBN 978-0-262-62107-6.

19 Christian Uhle and Jürgen Herre. Estimation of Tempo, Micro Time and Time Signature from Percussive Music. In *Proceedings of the 6th International Conference on Digital Audio Effects (DAFX)*, London, 2003.

20 William A Sethares. *Rhythm and transforms*. Springer, Berlin; London, 2007. ISBN 978-1-84628-640-7 1-84628-640-9 978-1-84628-639-1 1-84628-639-5. URL http://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&db=nlabk&AN=255666.

21 Paul Fraisse. Time and Rhythm Perception. In Edward C Carterette and Morton P Friedman, editors, *Perceptual Coding*, pages 203–254. Academic Press, January 1978. ISBN 978-0-12-161908-4. doi: 10.1016/B978-0-12-161908-4.50012-7. URL http://www.sciencedirect.com/science/article/pii/B9780121619084500127.

22 Richard Parncutt. Accents and Expression in Piano Performance. In Klaus W Niemöller, editor, *Perspektiven und Methoden einer Systemischen Musikwissenschaft*. Peter Lang, Frankfurt/Main, 2003.

23 Alf Gabrielsson. The Performance of Music. In Diana Deutsch, editor, *The Psychology of Music*. Academic Press, San Diego, CA, 2nd edition, 1999.

24 Bruno H Repp. On Determining the Basic Tempo of an Expressive Music Performance. *Psychology of Music*, 22(2):157–167, 1994.

25 Werner Goebl and Simon Dixon. Analysis of Tempo Classes in Performances of Mozart Sonatas. In *Proceedings of the 7th International Symposium on Systematic and Comparative Musicology (ISSCM), 3rd International Conference on Cognitive Musicology (ICCM)*, Jyväskylä, 2001.

26 Martin F McKinney and Dirk Moelants. Deviations from the Resonance Theory of Tempo Induction. In Richard Parncutt, A Kessler, and F Zimmer, editors, *Proceedings of the Conference on Interdisciplinary Musicology*, Graz, 2004.

27 Peter Desain. A (De)Composable Theory of Rhythm Perception. *Music Perception*, 9(4):439–454, 1992.

28 Vijay Iyer, Jeff Bilmes, Matt Wright, and David L Wessel. A Novel Representation for Rhythmic Structure. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 1–5, Thessaloniki, 1997. ICMA.

29 Ludger Hofmann-Engl. Rhythmic Similarity: A Theoretical and Empirical Approach. In *Proceedings of the 7th International Conference on Music Perception and Cognition (ICMPC)*, pages 564–567, Sydney, 2002.

30 Carl E Seashore. *Psychology of Music*. McGraw-Hill, New York, 1938.

31 Simon Dixon. Onset Detection Revisited. In *Proceedings of the 9th International Conference on Digital Audio Effects (DAFX)*, Montreal, September 2006.

32 Jonathan T Foote. Automatic Audio Segmentation Using a Measure of Audio Novelty. In *Proceedings of the International Conference on Multimedia and Expo (ICME)*, pages 452–455, New York, 2000.

33 Stephen Webley Hainsworth and Malcolm Macleod. Onset Detection in Musical Audio Signals. In *Proceedings of the International Computer Music Conference (ICMC)*, Singapore, 2003.

34 W Andrew Schloss. *On the Automatic Transcription of Percussive Music – From Acoustic Signal to High-Level Analysis*. PhD Thesis, Stanford University, Center for Computer Research in Music and Acoustics (CCRMA), Stanford, 1985.

35 Florian Eyben, Sebastian Böck, Björn Schuller, and Alex Graves. Universal Onset Detection with Bidirectional Long-Short Term Memory Neural Networks. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 589–594, Utrecht, Netherlands, 2010.

36 Sebastian Böck, Andreas Arzt, and Florian Krebs. Online Real-time Onset Detection with Recurrent Neural Networks. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, page 4, York, 2012.

37 Jan Schlüter and Sebastian Böck. Improved Musical Onset Detection with Convolutional Neural Networks. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6979–6983, Florence, Italy, May 2014. Institute of Electrical and Electronics Engineers (IEEE). doi: 10.1109/ICASSP.2014.6854953.

38 Eric D Scheirer. Tempo and beat analysis of acoustic musical signals. *Journal of the Acoustical Society of America (JASA)*, 103(1):588–601, 1998.

39 Anssi P Klapuri. Sound Onset Detection by Applying Psychoacoustic Knowledge. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Phoenix, 1999.

40 Chris Duxbury, Mark B Sandler, and Mike Davies. A Hybrid Approach to Musical Note Onset Detection. In *Proceedings Of The 5th International Conference On Digital Audio Effects*, Hamburg, 2002.

41 Ruohua Zhou and Joshua D Reiss. Music Onset Detection Combining Energy-Based and Pitch-Based Approaches. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Wien, September 2007.

42 Jean Laroche. Efficient Tempo and Beat Tracking in Audio Recordings. *Journal of the Audio Engineering Society (JAES)*, 51(4):226–233, 2003.

43 Chris Duxbury, Juan Pablo Bello, Mike Davies, and Mark B Sandler. Complex Domain Onset Detection for Musical Signals. In *Proceedings of the 6th International Conference on Digital Audio Effects (DAFX)*, London, 2003.

44 Juan Pablo Bello, Giuliano Monti, and Mark B Sandler. Phase-Based Note Onset Detection for Music Signals. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 5, Hong Kong, 2003.

45 Masataka Goto and Yoichi Muraoka. Music Understanding At The Beat Level – Real-time Beat Tracking For Audio Signals. In *Proceedings of the Workshop on Computational Auditory Scene Analysis (IJCAI)*, August 1995.

46 Axel Röbel. Onset Detection in Polyphonic Signals by means of Transient Peak Classification. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, 2005. URL http://www.music-ir.org/evaluation/mirex-results/articles/all/roebel.pdf.

47 Wei You and Roger Dannenberg. Polyphonic Music Note Onset Detection Using Semi-Supervised Learning. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 279–282, Vienna, Austria, 2007.

48 Alexandre Lacoste and Douglas Eck. A Supervised Classification Algorithm for Note Onset Detection. *EURASIP Journal on Advances in Signal Processing*, 2007(1):043745, 2007. ISSN 1687-6180. doi: 10.1155/2007/43745. URL http://asp.eurasipjournals.com/content/2007/1/043745.

49 Ruolun Liu, Niall John Lee Griffith, Jacqueline Walker, and Peter Murphy. Time Domain Note Average Energy Based Music Onset Detection. In *Proceedings of the Stockholm Music Acoustics Conference (SMAC)*, Stockholm, August 2003.

50 Hoon Heo, Dooyong Sung, and Kyogu Lee. Note Onset Detection based on Harmonic Cepstrum Regularity. In *Proceedings of the International Conference on Multimedia and Expo (ICME)*, pages 1–6, San Jose, CA, July 2013. Institute of Electrical and Electronics Engineers (IEEE). doi: 10.1109/ICME.2013.6607461. ISSN: 1945-788X.

51 Henrik von Coler and Alexander Lerch. CMMSD: A Data Set for Note-Level Segmentation of Monophonic Music. In *Proceedings of the AES 53rd International Conference on Semantic Audio*, London, UK, 2014. Audio Engineering Society (AES). URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2015/04/Coler_Lerch_2014_CMMSD.pdf.

52 Olivier Gillet and Gaël Richard. ENST-Drums: an Extensive Audio-Visual Database for Drum Signals Processing. In *ISMIR*, Victoria, 2006. URL http://ismir2006.ismir.net/PAPERS/ISMIR0627_Paper.pdf.

53 Ricard Marxer and Jordi Janer. Study of Regularizations and Constraints in NMF-based Drums Monaural Separation. In *Proceedings of the International Conference on Digital Audio Effects (DAFX)*, page 6, Maynooth, 2013.

54 Christian Dittmar and Daniel Gärtner. Real-time Transcription and Separation of Drum Recordings based on NMF Decomposition. In *Proceedings of the International Conference on Digital Audio Effects (DAFX)*, page 8, Erlangen, Germany, 2014.

55 Carl Southall, Chih-Wei Wu, Alexander Lerch, and Jason A Hockman. MDB Drums — An Annotated Subset of MedleyDB for Automatic Drum Transcription. In *Late Breaking Demo (Extended Abstract), Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Suzhou, 2017. International Society for Music Information Retrieval (ISMIR). URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2017/10/Wu-et-al_2017_MDB-Drums-An-Annotated-Subset-of-MedleyDB-for-Automatic-Drum-Transcription.pdf.

56 Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset. In *Proceedings of the International Conference on Learning Representations (ICLR)*, New Orleans, LA, January 2019. URL http://arxiv.org/abs/1810.12247.

57 George Tzanetakis. Tempo Extraction using Beat Histograms. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 2–3, 2005.

58 Anssi P Klapuri. Musical Meter Estimation and Music Transcription. In *Proceedings of the Cambridge Music Processing Colloquium*, Cambridge, 2003.

59 Jonathan T Foote and Shingo Uchihashi. The beat spectrum: a new approach to rhythm analysis. In *Proceedings of the International Conference on Multimedia and Expo (ICME)*, pages 881–884, Tokyo, August 2001.

60 George Tzanetakis and Perry Cook. Musical Genre Classification of Audio Signals. *Transactions on Speech and Audio Processing*, 10(5):293–302, July 2002. ISSN 1063-6676. doi: 10.1109/TSA.2002.800560. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1021072.

61 Juan José Burred and Alexander Lerch. Hierarchical Automatic Audio Signal Classification. *Journal of the Audio Engineering Society (JAES)*, 52(7/8):724–739, 2004. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2016/10/Burred-and-Lerch-2004-Hierarchical-Automatic-Audio-Signal-Classification.pdf.

62 Athanasios Lykartsis and Alexander Lerch. Beat Histogram Features for Rhythm-based Musical Genre Classification Using Multiple Novelty Functions. In *Proceedings of the International Conference on Digital Audio Effects (DAFX)*, Trondheim, Norway, 2015. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2015/12/DAFx-15_submission_42-1.pdf.

63 Matthew E P Davies, Philippe Hamel, Kazuyoshi Yoshii, and Masataka Goto. AutoMashUpper: Automatic Creation of Multi-Song Music Mashups. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(12):1726–1737, December 2014. ISSN 2329-9304. doi: 10.1109/TASLP.2014.2347135.

64 Len Vande Veire and Tijl De Bie. From Raw Audio to a Seamless Mix: Creating an Automated DJ System for Drum and Bass. *EURASIP Journal on Audio, Speech, and Music Processing*, 2018(1):13, September 2018. ISSN 1687-4722. doi: 10.1186/s13636-018-0134-8. URL https://doi.org/10.1186/s13636-018-0134-8.

65 Simon Dixon. A Beat Tracking System for Audio Signals. In *Proceedings of the Conference on Mathematical and Computational Methods in Music*, Vienna, 1999.

66 Fabien Gouyon and Perfecto Herrera. Determination of the Meter of Musical Audio Signals: Seeking Recurrances in Beat Segment Descriptors. In *Proceedings of the 114th Audio Engineering Society Convention*, Amsterdam, 2003. AES.

67 Paul E Allen and Roger B Dannenberg. Tracking Musical Beats in Real Time. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 140–143, Glasgow, September 1990.

68 Edward Wilson Large. Beat Tracking with a Nonlinear Oscillator. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, Montreal, August 1995.

69 David Rosenthal, Masataka Goto, and Yoichi Muraoka. Rhythm Tracking Using Multiple Hypotheses. In *Proceedings of the International Computer Music Conference (ICMC)*, Aarhus, September 1994.

70 Masataka Goto and Yoichi Muraoka. Beat Tracking based on Multiple-agent Architecture – A Real-time Beat Tracking System for Audio Signals. In *Proceedings of the Second International Conference on Multiagent Systems (ICMAS)*, 1996.

71 Masataka Goto and Yoichi Muraoka. Real-time Beat Tracking for Drumless Audio Signals: Chord Change Detection for Musical Decisions. *Speech Communication*, 27:311–335, 1999.

72 Simon Dixon. A Lightweight Multi-Agent Musical Beat Tracking System. In *Proceedings of the Pacific Rim International Conference on Artificial Intelligence (PRICAI)*, Melbourne, 2000.

73 Simon Dixon and Emilios Cambouropoulos. Beat Tracking with Musical Knowledge. In *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI)*, Berlin, August 2000.

74 Benoit Meudic. A Causal Algorithm for Beat Tracking. In *Proceedings of the 2nd International Conference on Understanding and Creating Music*, Caserta, November 2002.

75 Geoffroy Peeters. Time variable tempo detection and beat marking. In *Proceedings of the International Computer Music Conference (ICMC)*, Barcelona, September 2005.

76 Daniel P W Ellis. Beat Tracking by Dynamic Programming. *Journal of New Music Research*, 36(1):51–60, March 2007. ISSN 0929-8215. doi: 10.1080/09298210701653344. URL https://doi.org/10.1080/09298210701653344. Publisher: Routledge _eprint: https://doi.org/10.1080/09298210701653344.

77 Sebastian Böck, Florian Krebs, and Gerhard Widmer. Joint Beat and Downbeat Tracking with Recurrent Neural Networks. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 255–261. New York City, 2016.

78 Sebastian Böck, Matthew E P Davies, and Peter Knees. Multi-Task Learning of Tempo and Beat: Learning One to Improve the Other. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Delft, Netherlands, 2019.

79 Martin F McKinney and Dirk Moelants. Ambiguity in Tempo Perception: What Draws Listeners to Different Metrical Levels? *Music Perception*, 24(2):155–166, December 2006. ISSN 0730-7829. doi: 10.1525/mp.2006.24.2.155. URL https://doi.org/10.1525/mp.2006.24.2.155.

80 Fabien Gouyon, Anssi Klapuri, Simon Dixon, Miguel Alonso, George Tzanetakis, Christian Uhle, and Pedro Cano. An Experimental Comparison of Audio Tempo Induction Algorithms. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(5):1832–1844, September 2006. ISSN 1558-7924. doi: 10.1109/TSA.2005.858509.

81 André Holzapfel, Matthew E P Davies, José R Zapata, João Lobato Oliveira, and Fabien Gouyon. Selective Sampling for Beat Tracking Evaluation. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(9):2539–2548, November 2012. ISSN 1558-7924. doi: 10.1109/TASL.2012.2205244. Conference Name: IEEE Transactions on Audio, Speech, and Language Processing.

82 Simon Dixon. Evaluation of the Audio Beat Tracking System BeatRoot. *Journal of New Music Research*, 36(1):39–50, March 2007. ISSN 0929-8215. doi: 10.1080/09298210701653310. URL https://doi.org/10.1080/09298210701653310. Publisher: Routledge _eprint: https://doi.org/10.1080/09298210701653310.

83 Ali Taylan Cemgil, Bert Kappen, Peter Desain, and Henkjan Honing. On Tempo Tracking: Tempogram Representation and Kalman Filtering. *Journal of New Music Research*, 28(4):259–273, 2001.

84 Martin F McKinney, Dirk Moelants, Matthew E P Davies, and Anssi P Klapuri. Evaluation of Audio Beat Tracking and Music Tempo Extraction Algorithms. *Journal of New Music Research*, 36(1):1–16, March 2007. ISSN 0929-8215. doi: 10.1080/09298210701653252. URL https://doi.org/10.1080/09298210701653252. Publisher: Routledge _eprint: https://doi.org/10.1080/09298210701653252.

85 Matthew Davies and Sebastian Böck. Evaluating the Evaluation Measures for Beat Tracking. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Taipei, Taiwan, 2014.

86 Ugo Marchand and Geoffroy Peeters. The Extended Ballroom Dataset. In *Late Breaking Demo (Extended Abstract), Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 4, New York, 2016.

87 Hendrik Schreiber and Meinard Müller. A Crowdsourced Experiment for Tempo Estimation of Electonic Dance Music. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 7, Paris, France, 2018.

88 Graham Percival and George Tzanetakis. Streamlined Tempo Estimation Based on Autocorrelation and Cross-correlation With Pulses. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(12):1765–1776, December 2014. ISSN 2329-9304. doi: 10.1109/TASLP.2014.2348916. Conference Name: IEEE/ACM Transactions on Audio, Speech, and Language Processing.

89 Judith C Brown. Determination of the Meter of Musical Scores by Autocorrelation. *Journal of the Acoustical Society of America (JASA)*, 94(4):1953, 1993. ISSN 00014966. doi: 10.1121/1.407518. URL http://link.aip.org/link/JASMAN/v94/i4/p1953/s1&Agg=doi.

90 Petri Toiviainen and Tuomas Eerola. Autocorrelation in Meter Induction: The Role of Accent Structure. *Journal of the Acoustical Society of America (JASA)*, 119(2):1164–1170, 2006.

91 Bernardo Escalona-Espinosa. *Downbeat and Meter Estimation in Audio Signals*. Master's thesis, Technische Universität Hamburg-Harburg, 2008.

92 Stewart Macpherson. *Form in Music: With Special Reference to the Designs of Instrumental Music*. Joseph Williams, Ltd., London, UK, 1915. URL http://archive.org/details/forminmusicwiths00macp.

93 Ebenezer Prout. *Musical Form*. Augener, London, UK, 1897. URL http://archive.org/details/musicalform00prouuoft.

94 Glenn Spring and Jere Hutcheson. *Musical Form and Analysis: Time, Pattern, Proportion*. Waveland Press, Long Grove, IL, 2013. ISBN 978-1-4786-0722-9. OCLC: 882602291.

95 Caroline Palmer. Music Performance. *Annual Review of Psychology*, 48:115–138, 1997.

96 Jouni Paulus, Meinard Muller, and Anssi P Klapuri. State of the Art Report: Audio-Based Music Structure Analysis. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 625–636, Utrecht, Netherlands, 2010. URL http://ismir2010.ismir.net/proceedings/ismir2010-107.pdf.

97 Oriol Nieto, Gautham J Mysore, Cheng-i Wang, Jordan B L Smith, Jan Schlüter, Thomas Grill, and Brian McFee. Audio-Based Music Structure Analysis: Current Trends, Open Challenges, and Applications. *Transactions of the International Society for Music Information Retrieval (TISMIR)*, 3(1):246–263, 2020.

98 Masataka Goto. A Chorus Section Detection Method for Musical Audio Signals and its Application to a Music Listening Station. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(5):1783–1794, September 2006. ISSN 1558-7924. doi: 10.1109/TSA.2005.863204.

99 Jean-Julien Aucouturier and Mark Sandler. Finding Repeating Patterns in Acoustic Musical Signals: Applications for Audio Thumbnailing. Audio Engineering Society, June 2002. URL https://www.aes.org/e-lib/browse.cfm?elib=11163.

100 Pierre Soille. Erosion and Dilation. In *Morphological Image Analysis: Principles and Applications*, pages 63–103. Springer, Berlin, Heidelberg, 2004. ISBN 978-3-662-05088-0. URL https://doi.org/10.1007/978-3-662-05088-0_3.

101 Mark Levy and Mark Sandler. Structural Segmentation of Musical Audio by Constrained Clustering. *Transactions on Audio, Speech, and Language Processing*, 16(2):318–326, February 2008. ISSN 1558-7916. doi: 10.1109/TASL.2007.910781. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4432648.

102 Brian McFee and Daniel P W Ellis. Analyzing Song Structure with Spectral Clustering. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 6, Taipei, Taiwan, 2014.

103 Roger B Dannenberg and Masataka Goto. Music Structure Analysis from Acoustic Signals. In David Havelock, Sonoko Kuwano, and Michael Vorländer, editors, *Handbook of Signal Processing in Acoustics*, pages 305–331. Springer, New York, NY, 2008. ISBN 978-0-387-30441-0. URL https://doi.org/10.1007/978-0-387-30441-0_21.

104 Katherine M Kinnaird. Aligned Hierarchies: A Multi-Scale Structure-Based Representation for Music-Based Data Streams. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 7, New York, 2016.

105 Brian McFee, Oriol Nieto, Morwaread M Farbood, and Juan Pablo Bello. Evaluating Hierarchical Structure in Music Annotations. *Frontiers in Psychology*, 2017. ISSN 1664-1078. doi: 10.3389/fpsyg.2017.01337. URL https://internal-journal.frontiersin.org/articles/10.3389/fpsyg.2017.01337/full. Publisher: Frontiers.

106 Katherine M Kinnaird and Brian McFee. Automatic Hierarchy Expansion for Improved Structure and Chord Evaluation. *Transactions of the International Society for Music Information Retrieval*, 4(1):81–92, June 2021. ISSN 2514-3298. doi: 10.5334/tismir.71. URL http://transactions.ismir.net/article/10.5334/tismir.71/. Number: 1 Publisher: Ubiquity Press.

107 Douglas Turnbull, Gert Lanckriet, Elias Pampalk, and Masataka Goto. A Supervised Approach for Detecting Boundaries in Music Using Difference Features and Boosting. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 5, Vienna, Austria, 2007.

108 Hanna Lukashevich. Towards Quantitative Measures of Evaluating Song Segmentation. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 6, Philadelphia, PA, 2008.

109 Masataka Goto. RWC Music Database: Popular, Classical, and Jazz Music Databases. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 2, Paris, 2002.

110 Frédéric Bimbot, Gabriel Sargent, Emmanuel Deruty, Corentin Guichaoua, and Emmanuel Vincent. Semiotic Description of Music Structure: An Introduction to the Quaero/Metiss Structural Annotations. Audio Engineering Society, January 2014. URL https://www.aes.org/e-lib/online/browse.cfm?elib=17115.

111 Matthias Mauch, Chris Cannam, Matthew E P Davies, Simon Dixon, Christopher A Harte, S Kolozali, and Dan Tidhar. OMRAS2 Metadata Project 2009. In *Late Breaking Demo (Extended Abstract), Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 1, Kobe, 2009.

112 Oriol Nieto and Juan Pablo Bello. Systematic Exploration of Computational Music Structure Research. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 547–553, 2016. URL http://dblp.uni-trier.de/search?q=oriol\LY1\textbackslash%20nieto.

113 Jordan B L Smith, J Ashley Burgoyne, Ichiro Fujinaga, David De Roure, and J Stephen Downie. Design and Creation of a Large-Scale Databse of

Structural Annotations. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 7, Miami, FL, 2011.

114 Chih-Wei Wu, Christian Dittmar, Carl Southall, Richard Vogl, Gerhard Widmer, Jason A Hockman, Meinard Muller, and Alexander Lerch. A Review of Automatic Drum Transcription. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26(9):1457–1483, 2018. ISSN 2329-9290. doi: 10.1109/TASLP.2018.2830113. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2018/05/Wu-et-al.-2018-A-review-of-automatic-drum-transcription.pdf.

115 Fabien Gouyon, François Pachet, and Olivier Delerue. On the Use of Zero-Crossing Rate for an Application of Classification of Percussive Sounds. In *Proceedings of the Conference on Digital Audio Effects (DAFX)*, Verona, 2000. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.18.7006&rep=rep1&type=pdf.

116 Perfecto Herrera, Alexandre Yeterian, and Fabien Gouyon. Automatic Classification of Drum Sounds : A Comparison of Feature Selection Methods and Classification Techniques. In *Proceedings of the 2nd International Conference on Music and Artificial Intelligence (ICMAI)*, 2002.

117 Perfecto Herrera, Amaury Dehamel, and Fabien Gouyon. Automatic Labeling of Unpitched Percussion Sounds. In *Proceedings of the 114th Audio engineering Society Convention*, Amsterdam, 2003.

118 Dirk Van Steelant, Koen Tanghe, Sven Degroeve, Bernard De Baets, Marc Leman, and Jean-Pierre Martens. Classification of Percussive Sounds Using Support Vector Machines. In *Proceedings of Annual Machine Learning Conference of Belgium and The Netherlands (BENELEARN)*, page 8, 2004. URL http://hdl.handle.net/1854/LU-219378.

119 Pierre Roy, François Pachet, and Sergio Krakowski. Improving the Classification of Percussive Sounds with Analytical Features: A Case Study. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 5, Vienna, Austria, 2007.

120 Derry FitzGerald and Bob Lawlor. Sub-band Independent Subspace Analysis For Drum Transcription. In *Proceedings of the International Conference on Digital Audio Effects (DAFX)*, Hamburg, Germany, 2002.

121 Olivier Gillet and Gaël Richard. Automatic Transcription of Drum Loops. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages iv–269–iv–272, vol. 4, May 2004. doi: 10.1109/ICASSP.2004.1326815. 00062.

122 Jouni Paulus and Tuomas Virtanen. Drum Transcription with Non-Negative Spectrogram Factorisation. In *Proceedings of the 13th European Signal Processing Conference (EUSIPCO)*, page 4, Talya, 2005. URL http://www.eurasip.org/Proceedings/Eusipco/Eusipco2005/defevent/papers/cr1410.pdf.

123 Umut Simsekli, Antti Jylhä, Cumhur Erkut, and A. Taylan Cemgil. Real-Time Recognition of Percussive Sounds by a Model-Based Method. *EURASIP Journal on Advances in Signal Processing*, 2011. ISSN 16876172. URL https://www.proquest.com/docview/856980321/abstract/4D4B2515F3F94654PQ/1. Place: New York, Netherlands Publisher: Springer Nature B.V.

124 Aymeric Zils, François Pachet, Olivier Delerue, and Fabien Gouyon. Automatic Extraction of Drum Tracks from Polyphonic Music Signals. In *Proceedings of the International Conference on Web Delivering of Music (WEDELMUSIC)*, pages 179–183, Darmstadt, December 2002. doi: 10.1109/WDM.2002.1176209.

125 Vegard Sandvold, Fabien Gouyon, and Perfecto Herrera. Percussion Classification in Polyphonic Audio Recordings Using Localized Sound Models. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 4, Barcelona, Spain, 2004.

126 Kazuyoshi Yoshii, Masataka Goto, and Hiroshi G Okuno. Drum Sound Recognition for Polyphonic Audio Signals by Adaptation and Matching of Spectrogram Templates With Harmonic Structure Suppression. *IEEE Transactions on Audio, Speech and Language Processing*, 15(1):333–345, January 2007. ISSN 1558-7916. doi: 10.1109/TASL.2006.876754. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4032798.

127 Olivier Gillet and Gaël Richard. Transcription and Separation of Drum Signals From Polyphonic Music. *Transactions on Audio, Speech, and Language Processing*, 16(3):529–540, March 2008. ISSN 1558-7916. doi: 10.1109/TASL.2007.914120. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4443887.

128 David S Alves, Jouni Paulus, and José Fonseca. Drum Transcription from Multichannel Recordings with Non-Negative Matrix Factorization. In *Proceedings of the European Signal Processing Conference (EUSIPCO)*, Glasgow, 2009. URL http://www.researchgate.net/publication/229044064_Drum_transcription_from_multichannel_recordings_with_non-negative_matrix_factorization/file/60b7d52c563d962996.pdf.

129 Jouni Paulus and Anssi Klapuri. Drum Sound Detection in Polyphonic Music with Hidden Markov Models. *EURASIP Journal on Audio, Speech, and Music Processing*, 2009. ISSN 16874714. URL https://www.proquest.com/docview/855531882/abstract/8C806C20A2794B58PQ/1. Place: New York, Netherlands Publisher: Springer Nature B.V.

130 Emmanouil Benetos, Sebastian Ewert, and Tillman Weyde. Automatic Transcription of Pitched and Unpitched Sounds from Polyphonic Music. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3107–3111, Florence, Italy, May 2014. doi: 10.1109/ICASSP.2014.6854172. ISSN: 2379-190X.

131 Axel Röbel, Jordi Pons, Marco Liuni, and Mathieu Lagrange. On Automatic Drum Transcription Using Non-negative Matrix Deconvolution and Itakura Saito Divergence. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 414–418, South Brisbane, Australia, April 2015. doi: 10.1109/ICASSP.2015.7178002.

132 Chih-Wei Wu and Alexander Lerch. Drum Transcription using Partially Fixed Non-Negative Matrix Factorization. In *Proceedings of the European Signal Processing Conference (EUSIPCO)*, Nice, 2015. EURASIP. doi: 10.1109/EUSIPCO.2015.7362590. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2015/09/Wu_Lerch_2015_Drum%20Transcription%20using%20Partially%20Fixed%20Non-Negative%20Matrix%20Factorization.pdf.

133 Chih-Wei Wu and Alexander Lerch. Drum Transcription using Partially Fixed Non-Negative Matrix Factorization With Template Adaptation. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Malaga, 2015. ISMIR. doi: 10.5281/zenodo.1417839. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2015/10/Wu_Lerch_2015_Drum-Transcription-using-Partially-Fixed-Non-Negative-Matrix-Factorization-With.pdf.

134 Richard Vogl, Matthias Dorfer, and Peter Knees. Drum Transcription from Polyphonic Music with Recurrent Neural Networks. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 201–205, March 2017. doi: 10.1109/ICASSP.2017.7952146. ISSN: 2379-190X.

135 Carl Southall, Ryan Stables, and Jason Hockman. Automatic Drum Transcription for Polyphonic Recordings Using Soft Attention Mechanisms and Convolutional Neural Networks. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Suzhou, China, 2017. ISBN 978-0-692-75506-8. URL https://www.open-access.bcu.ac.uk/4747/. Conference Name: International Society of Music Information Retrieval Conference Meeting Name: International Society of Music Information Retrieval Conference.

136 Len Vande Veire, Cedric De Boom, and Tijl De Bie. Sigmoidal NMFD: Convolutional NMF with Saturating Activations for Drum Mixture Decomposition. *Electronics*, 10(3):284, January 2021. doi: 10.3390/electronics10030284. URL https://www.mdpi.com/2079-9292/10/3/284. Number: 3 Publisher: Multidisciplinary Digital Publishing Institute.

137 Richard Vogl, Gerhard Widmer, and Peter Knees. Towards Multi-Instrument Drum Transcription. In *Proceedings of the International Conference on Digital Audio Effects (DAFX)*, Aveiro, Portugal, October 2018. URL http://arxiv.org/abs/1806.06676.

138 Yu Wang, Justin Salamon, Mark Cartwright, Nicholas J Bryan, and Juan Pablo Bello. Few-Shot Drum Transcription in Polyphonic Music. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Montreal (Virtual Conference), Canada, August 2020. URL http://arxiv.org/abs/2008.02791.

139 Céline Jacques and Axel Roebel. Data Augmentation for Drum Transcription with Convolutional Neural Networks. In *2019 27th European Signal Processing Conference (EUSIPCO)*, pages 1–5, September 2019. doi: 10.23919/EUSIPCO.2019.8902980. ISSN: 2076-1465.

140 Chih-Wei Wu and Alexander Lerch. Automatic Drum Transcription using the Student-Teacher Learning Paradigm with Unlabeled Music Data. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Suzhou, 2017. International Society for Music Information Retrieval (ISMIR). doi: 10.5281/zenodo.1415904. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2017/07/Wu_Lerch_2017_Automatic-drum-transcription-using-the-student-teacher-learning-paradigm-with.pdf.

141 Chih-Wei Wu and Alexander Lerch. From Labeled to Unlabeled Data – On the Data Challenge in Automatic Drum Transcription. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Paris, 2018. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2018/06/Wu-and-Lerch-From-Labeled-to-Unlabeled-Data-On-the-Data-Chal.pdf.

142 Keunwoo Choi and Kyunghyun Cho. Deep Unsupervised Drum Transcription. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Delft, Netherlands, 2019.

143 I-Chieh Wei, Chih-Wei Wu, and Li Su. Improving Automatic Drum Transcription Using Large-Scale Audio-to-Midi Aligned Data. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 246–250, June 2021. doi: 10.1109/ICASSP39728.2021.9414409. ISSN: 2379-190X.

144 Lee Callender, Curtis Hawthorne, and Jesse Engel. Improving Perceptual Quality of Drum Transcription with the Expanded Groove MIDI Dataset. *arXiv:2004.00188 [cs]*, December 2020. URL http://arxiv.org/abs/2004.00188.

145 Mickael Zehren, Marco Alunno, and Paolo Bientinesi. ADTOF: A Large Dataset of Non-synthetic Music for Automatic Drum Transcription. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Online, November 2021. doi: 10.5281/zenodo.5624527. URL http://arxiv.org/abs/2111.11737.

10

Alignment

The alignment of two sequences results in a direct mapping of every time instant in one sequence to at least one time instant in the other sequence by minimizing some predefined alignment cost. Computing the alignment of two sequences makes most sense if the two sequences are similar yet not identical, such as two music performances of the same piece with varying tempo. The target mapping between two sequences could be visualized like in Figure 10.1.

The results of the alignment can be used for various Audio Content Analysis (ACA) tasks. They can, for example, be used for estimating the overall similarity between two audio sequences or to generate an onset-synchronized score visualization after aligning an audio file with its musical score.

While sequence alignment algorithms are usually independent of the application domain, this chapter looks at music alignment, specifically audio-to-audio and audio-to-score alignment algorithms. This chapter first introduces a standard task-agnostic approach to sequence alignment (or path finding), the Dynamic Time Warping (DTW) algorithm, and then provides an overview of standard approaches to audio-to-audio alignment and audio-to-score alignment. The Viterbi algorithm (see Section 7.6.2) is a closely related approach to path finding.

10.1 Dynamic Time Warping

The objective of *DTW* is to align or to synchronize two sequences of different length [1]. The algorithm assumes that both sequences start at their first index and end at their last index, which means it is not suitable for subsequence alignment in its default form. Furthermore, it assumes that a mapping sequence – the *path* – cannot jump over multiple indices and that the mapped indices can never decrease. This section will first attempt to introduce DTW in an abstract way before providing a step-by-step example that should clarify the practical use.

An Introduction to Audio Content Analysis: Music Information Retrieval Tasks & Applications,
Second Edition. Alexander Lerch.
© 2023 The Institute of Electrical and Electronics Engineers, Inc. Published 2023 by John Wiley & Sons, Inc.

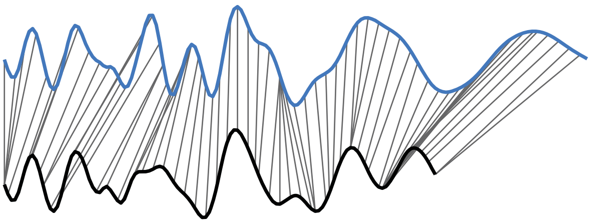


Figure 10.1 Visualization of the mapping of two similar sequences to each other after alignment.

Repo 1: plotSequenceAlignment.m

Given two sequences $A(n_\mathrm{A})$ with $n_\mathrm{A} \in [0; \mathcal{N}_\mathrm{A} - 1]$ and $B(n_\mathrm{B})$ with $n_\mathrm{B} \in [0; \mathcal{N}_\mathrm{B} - 1]$, the first step is the computation of a pairwise distance between $A(n_\mathrm{A})$ and $B(n_\mathrm{B})$. This results in the *distance matrix* $\boldsymbol{D}_\mathrm{AB}(n_\mathrm{A}, n_\mathrm{B})$. If the two sequences are identical, the result is very similar (albeit inverted) to the Self Similarity Matrix (SSM) introduced in Section 9.7.1.

The goal of the algorithm is to find the specific *alignment path* from $\boldsymbol{D}_\mathrm{AB}(0, 0)$ (the start of both sequences) to $\boldsymbol{D}_\mathrm{AB}(\mathcal{N}_\mathrm{A} - 1, \mathcal{N}_\mathrm{B} - 1)$ (the end of both sequences) which minimizes the overall accumulated distance. The path minimizing the accumulated distance is the most likely alignment path (also *warping path*) between the two sequences. Figure 10.2 shows two example sequences, the corresponding distance matrix, and the resulting alignment path between the two sequences computed with standard DTW as described below.

The alignment path will be referred to as $\boldsymbol{p}(n_\mathrm{P})$ and $n_\mathrm{P} \in [0; \mathcal{N}_\mathrm{P} - 1]$; it is a direct measure of how one sequence has to be warped (scaled in time) to give the best fit to the other sequence. Each path entry is a matrix index in the range $([0, \mathcal{N}_\mathrm{A} - 1]; [0, \mathcal{N}_\mathrm{B} - 1])$.

As indicated above, the following constraints apply to the path:

- *Boundaries*: the path has to start at the first index of both sequences and has to end at the end of both sequences, meaning that it covers both entire sequences from beginning to end:

$$\boldsymbol{p}(0) = [0, 0], \tag{10.1}$$

$$\boldsymbol{p}(\mathcal{N}_\mathrm{P} - 1) = [\mathcal{N}_\mathrm{A} - 1, \mathcal{N}_\mathrm{B} - 1]. \tag{10.2}$$

- *Causality*: the path can only move forward through both sequences, meaning that it is not allowed to “go back in time”:

$$n_\mathrm{A}\big|_{\boldsymbol{p}(n_\mathrm{P})} \leq n_\mathrm{A}\big|_{\boldsymbol{p}(n_\mathrm{P}+1)}, \tag{10.3}$$

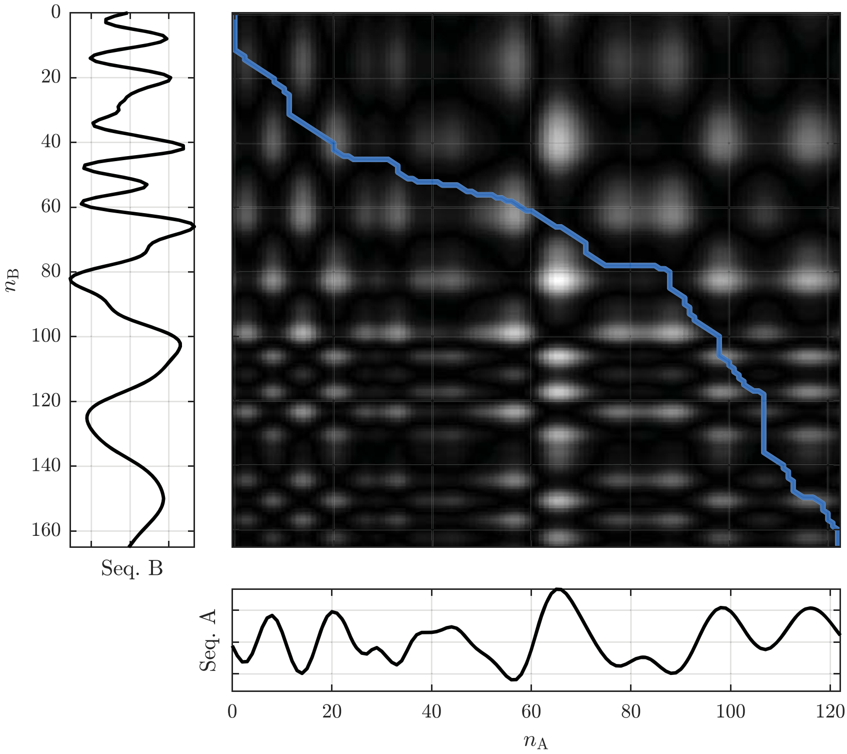
$$n_\mathrm{B}\big|_{\boldsymbol{p}(n_\mathrm{P})} \leq n_\mathrm{B}\big|_{\boldsymbol{p}(n_\mathrm{P}+1)}, \tag{10.4}$$

$$(n_\mathrm{A} + n_\mathrm{B})\big|_{\boldsymbol{p}(n_\mathrm{P})} < (n_\mathrm{A} + n_\mathrm{B})\big|_{\boldsymbol{p}(n_\mathrm{P}+1)}. \tag{10.5}$$

- *Continuity*: no index n_A or n_B can be omitted, meaning that the path is not allowed to jump through either sequence:

$$n_\mathrm{A}\big|_{\boldsymbol{p}(n_\mathrm{P}+1)} \leq (n_\mathrm{A} + 1)\big|_{\boldsymbol{p}(n_\mathrm{P})}, \tag{10.6}$$

$$n_\mathrm{B}\big|_{\boldsymbol{p}(n_\mathrm{P}+1)} \leq (n_\mathrm{B} + 1)\big|_{\boldsymbol{p}(n_\mathrm{P})}. \tag{10.7}$$



Repo 1: plotDtwPath.m

Figure 10.2 Distance matrix and alignment path for two example sequences; dark entries indicate a small distance.

These path restrictions result in a theoretical maximum path length of

$$\mathcal{N}_{\mathrm{P,max}} = \mathcal{N}_\mathrm{A} + \mathcal{N}_\mathrm{B} - 2 \tag{10.8}$$

when the path runs along the edges of the distance matrix[1] and a minimum path length of

$$\mathcal{N}_{\mathrm{P,min}} = \max\left(\mathcal{N}_\mathrm{A}, \mathcal{N}_\mathrm{B}\right) \tag{10.9}$$

when the path runs on the matrix diagonal for as long as possible.

In order to find the optimal alignment path, the concept of “cost” is used. The cost of a path $\boldsymbol{p}_j$ is determined by accumulating the values of the distance matrix at all path points:

$$\mathfrak{C}_\mathrm{AB}(j) = \sum_{n_\mathrm{P}=0}^{\mathcal{N}_\mathrm{P}-1} \boldsymbol{D}\left(\boldsymbol{p}_j(n_\mathrm{P})\right). \tag{10.10}$$

---

[1] The −2 originates in the automatic avoidance of the corner of the distance matrix.

The goal is, therefore, to identify the optimal alignment path which is minimizing the overall cost:

$$\mathfrak{C}_{\mathrm{AB},min} = \min_{\forall j} \left(\mathfrak{C}_{\mathrm{AB}}(j)\right), \tag{10.11}$$

$$j_{\mathrm{opt}} = \underset{\forall j}{\operatorname{argmin}} \left(\mathfrak{C}_{\mathrm{AB}}(j)\right). \tag{10.12}$$

The optimal path can theoretically be found by computing all possible paths through the matrix $\boldsymbol{D}$ and determining the path with the lowest overall cost, however, this brute-force approach will be very inefficient for long sequences.

Dynamic Programming (DP) techniques allow to avoid this brute force approach as the best global solution can be computed more efficiently. As an intermediate result, the *cost matrix* $\boldsymbol{C}_{\mathrm{AB}}$ is introduced; it has the same dimensions as the distance matrix, but each matrix element contains the accumulated overall cost of the best (i.e. least expensive) path to this specific matrix element.

The cost matrix can be computed iteratively by

$$\boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}, n_{\mathrm{B}}) = \boldsymbol{D}_{\mathrm{AB}}(n_{\mathrm{A}}, n_{\mathrm{B}}) + \min \begin{cases} \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}} - 1, n_{\mathrm{B}} - 1) \\ \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}} - 1, n_{\mathrm{B}}) \\ \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}, n_{\mathrm{B}} - 1) \end{cases}. \tag{10.13}$$

Figure 10.3 plots both the distance matrix and the corresponding cost matrix for the example sequences from Figure 10.2.

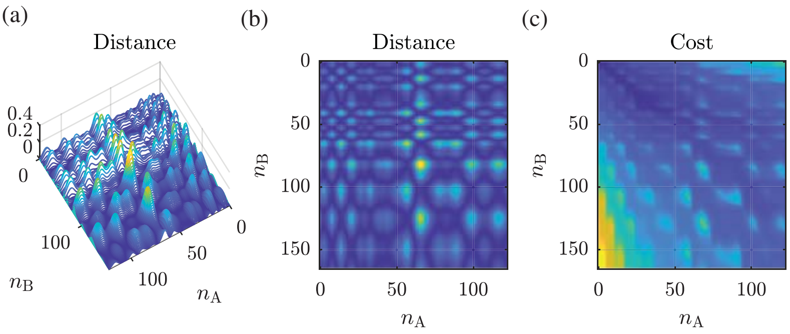
The first entry of the cost matrix is initialized with

$$\boldsymbol{C}_{\mathrm{AB}}(0,0) = \boldsymbol{D}_{\mathrm{AB}}(0,0). \tag{10.14}$$

The alignment path restrictions given above make the computation of both the first row and the first column of the cost matrix trivial:

$$\boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}, 0) = \boldsymbol{D}_{\mathrm{AB}}(n_{\mathrm{A}}, 0) + \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}} - 1, 0), \tag{10.15}$$

$$\boldsymbol{C}_{\mathrm{AB}}(0, n_{\mathrm{B}}) = \boldsymbol{D}_{\mathrm{AB}}(0, n_{\mathrm{B}}) + \boldsymbol{C}_{\mathrm{AB}}(0, n_{\mathrm{B}} - 1). \tag{10.16}$$



Repo 1: plotDtwCost.m

Figure 10.3 Distance matrix (a and b in two different visualizations) and corresponding cost matrix (c).

This can also be interpreted as initializing the distance matrix for indices smaller than 0 with

$$\begin{aligned}
&\boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}, -1) = \infty, \\
&\boldsymbol{C}_{\mathrm{AB}}(-1, n_{\mathrm{B}}) = \infty,
\end{aligned}$$

and applying Eq. (10.13).

Each cost matrix element contains the minimum cost to reach that element. However, knowledge of the direction from which this element was reached is crucial to finding a path. Therefore, during the calculation of the cost matrix, the indices of the preceding cell with the minimum cost have to be remembered for each matrix element. The optimal path can then be traced back from the any matrix element to the beginning.

The complete iterative algorithm can thus formally be summarized by

- *Initialization*:

$$\boldsymbol{C}_{\mathrm{AB}}(0,0) = \boldsymbol{D}_{\mathrm{AB}}(0,0), \tag{10.17}$$

$$\boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}, -1) = \infty, \tag{10.18}$$

$$\boldsymbol{C}_{\mathrm{AB}}(-1, n_{\mathrm{B}}) = \infty. \tag{10.19}$$

- *Recursion*:

$$\boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}, n_{\mathrm{B}}) = \boldsymbol{D}_{\mathrm{AB}}(n_{\mathrm{A}}, n_{\mathrm{B}}) + \min \begin{cases} \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}} - 1, n_{\mathrm{B}} - 1) \\ \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}} - 1, n_{\mathrm{B}}) \\ \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}, n_{\mathrm{B}} - 1) \end{cases}, \tag{10.20}$$

$$j = \operatorname{argmin} \begin{cases} \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}} - 1, n_{\mathrm{B}} - 1) \\ \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}} - 1, n_{\mathrm{B}}) \\ \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}, n_{\mathrm{B}} - 1) \end{cases}, \tag{10.21}$$

$$\Delta\boldsymbol{p}(n_{\mathrm{A}}, n_{\mathrm{B}}) = \begin{cases} [-1,-1] & \text{if } j = 0 \\ [-1,0] & \text{if } j = 1 \\ [0,-1] & \text{if } j = 2 \end{cases}. \tag{10.22}$$

- *Termination*:

$$n_{\mathrm{A}} = \mathcal{N}_{\mathrm{A}} - 1 \wedge n_{\mathrm{B}} = \mathcal{N}_{\mathrm{B}} - 1. \tag{10.23}$$

- *Path backtracking*:

$$\begin{aligned}
&\boldsymbol{p}(n_{\mathrm{P}}) = \boldsymbol{p}(n_{\mathrm{P}} + 1) + \Delta\boldsymbol{p}\big(\boldsymbol{p}(n_{\mathrm{P}} + 1)\big), \\
&\quad n_{\mathrm{P}} = \mathcal{N}_{\mathrm{P}} - 2, \mathcal{N}_{\mathrm{P}} - 3, \ldots, 0.
\end{aligned} \tag{10.24}$$

Listing 10.1 shows an example implementation of Eqs. (10.20)–(10.24).

It can occur that a matrix has more than one optimal path in the case the cost is identical for two different routes through the matrix. As these paths are equivalent in terms of cost, there is no general preference as to which to choose.

```
% init directions for back-tracking [diag, vert, hori]
iDec    = [-1 -1; -1 0; 0 -1];

% recursion
for n_A = 2:size(D, 1)
    for n_B = 2:size(D, 2)
        % find preceding min (diag, column, row)
        [fC_min, DeltaP(n_A, n_B)] = min([C(n_A-1, n_B-1), ...
                                           C(n_A-1, n_B), ...
                                           C(n_A, n_B-1)]);
        C(n_A, n_B) = D(n_A, n_B) + fC_min;
    end
end

% traceback
p = size(D);  % start with the last element
n = [size(D, 1), size(D, 2)]; %[n_A, n_B];
while ((n(1) > 1) || (n(2) > 1))
    n = n + iDec(DeltaP(n(1),n(2)), :);

    % update path (final length unknown)
    p = [n; p];
end
```

Repo 2: ToolSimpleDtw.m
Repo 3: ToolSimpleDtw.py

Listing 10.1 Implementation of DTW.

Note that the distance matrix can be replaced by a *similarity matrix*; the algorithm will remain the same although the cost matrix (as introduced below) will have to be replaced by a *likelihood matrix* and some algorithmic details will have to be modified in other places as well, for instance, replacing the min operation by a max operation.

10.1.1 Example

A short example is presented here in order to allow a better understanding of the initially rather abstract concept of DTW. The two (one-dimensional) input sequences are

$$A = [1,\ 2,\ 3,\ 0],$$
$$B = [1,\ 0,\ 2,\ 3,\ 1],$$

and we will simply use the magnitude of the difference as the distance measure. The distance matrix is then

$$\boldsymbol{D}_{\mathrm{AB}} = \begin{bmatrix} 0 & 1 & 2 & 1 \\ 1 & 2 & 3 & 0 \\ 1 & 0 & 1 & 2 \\ 2 & 1 & 0 & 3 \\ 0 & 1 & 2 & 1 \end{bmatrix}, \qquad (10.25)$$

and the corresponding cost matrix is

$$\boldsymbol{C}_{\mathrm{AB}} = \begin{bmatrix} 0 & 1 & 3 & 4 \\ 1 & 2 & 4 & 3 \\ 2 & 1 & 2 & 4 \\ 4 & 1 & 1 & 4 \\ 4 & 3 & 3 & 2 \end{bmatrix}. \tag{10.26}$$

During the calculation of the cost matrix, we also memorized the direction of lowest cost for each matrix element:

$$\begin{bmatrix} \cdot & \leftarrow & \leftarrow & \leftarrow \\ \uparrow & \nwarrow & \nwarrow & \nwarrow \\ \uparrow & \nwarrow & \leftarrow & \leftarrow \\ \uparrow & \uparrow & \nwarrow & \leftarrow \\ \uparrow & \uparrow & \uparrow & \nwarrow \end{bmatrix}$$

in order to be able to backtrack the optimal path through the distance matrix:

$$\boldsymbol{D}_{\mathrm{AB}} = \begin{bmatrix} 0 & 1 & 2 & 1 \\ 1 & 2 & 3 & 0 \\ 1 & 0 & 1 & 2 \\ 2 & 1 & 0 & 3 \\ 0 & 1 & 2 & 1 \end{bmatrix}. \tag{10.27}$$

10.1.2 Common Variants

The standard DTW algorithm as described above is sometimes modified; the reasons for this modification can be either the adaption to specific use cases or the optimization of its workload requirements. Several such modifications are described in detail by Müller [2].

In order to favor vertical, horizontal, or diagonal path movement, weighting factors can be applied to Eq. (10.13)

$$\boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}, n_{\mathrm{B}}) = \min \begin{cases} \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}} - 1, n_{\mathrm{B}} - 1) & + \quad \lambda_{\mathrm{d}} \cdot \boldsymbol{D}_{\mathrm{AB}}(n_{\mathrm{A}}, n_{\mathrm{B}}) \\ \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}} - 1, n_{\mathrm{B}}) & + \quad \lambda_{\mathrm{v}} \cdot \boldsymbol{D}_{\mathrm{AB}}(n_{\mathrm{A}}, n_{\mathrm{B}}) \\ \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}, n_{\mathrm{B}} - 1) & + \quad \lambda_{\mathrm{h}} \cdot \boldsymbol{D}_{\mathrm{AB}}(n_{\mathrm{A}}, n_{\mathrm{B}}) \end{cases}. \tag{10.28}$$

In the default DTW approach, all these weights had been set to 1. An alternative set of weights is

$$\lambda_{\mathrm{d}} = 2,$$
$$\lambda_{\mathrm{v}} = 1,$$
$$\lambda_{\mathrm{h}} = 1$$

to prevent the implicit preference of the diagonal since one diagonal step corresponds to one horizontal plus one vertical step.

Instead of forcing the algorithm to only increment each index by one, it is conceivable to loosen this constraint to allow for larger step sizes or jumps. The arguments of the minimum operation of Eq. (10.13) could, for example, be replaced by

$$\min \begin{cases} \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}-1, n_{\mathrm{B}}-1) \\ \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}-2, n_{\mathrm{B}}-1) \\ \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}-1, n_{\mathrm{B}}-2) \end{cases} \tag{10.29}$$

which constrains the slope of the warping path to avoid the path containing many consecutive horizontal or vertical steps. This modification will only work if the sequence lengths $\mathcal{N}_{\mathrm{A}}$ and $\mathcal{N}_{\mathrm{B}}$ differ not by more than a factor of 2. Another alternative enforcing the alignment of all elements of both sequences is to replace the arguments of the minimum operation of Eq. (10.13) by

$$\min \begin{cases} \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}-1, n_{\mathrm{B}}-1) \\ \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}-2, n_{\mathrm{B}}-1) + \boldsymbol{D}_{\mathrm{AB}}(n_{\mathrm{A}}-1, n_{\mathrm{B}}) \\ \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}-3, n_{\mathrm{B}}-1) + \boldsymbol{D}_{\mathrm{AB}}(n_{\mathrm{A}}-1, n_{\mathrm{B}}) + \boldsymbol{D}_{\mathrm{AB}}(n_{\mathrm{A}}-2, n_{\mathrm{B}}) \\ \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}-1, n_{\mathrm{B}}-2) + \boldsymbol{D}_{\mathrm{AB}}(n_{\mathrm{A}}, n_{\mathrm{B}}-1) \\ \boldsymbol{C}_{\mathrm{AB}}(n_{\mathrm{A}}-1, n_{\mathrm{B}}-3) + \boldsymbol{D}_{\mathrm{AB}}(n_{\mathrm{A}}, n_{\mathrm{B}}-1) + \boldsymbol{D}_{\mathrm{AB}}(n_{\mathrm{A}}, n_{\mathrm{B}}-2). \end{cases} \tag{10.30}$$

10.1.3 Optimizations

If the two sequences to be aligned are long, the size of the distance matrix increases drastically as the number of matrix elements is the multiplication of the length of the sequences $\mathcal{N}_{\mathrm{A}} \cdot \mathcal{N}_{\mathrm{B}}$. This results in both high memory requirements and a large overall number of operations. The effects can be alleviated by using different approaches to optimization.

One simple approach to reduce the amount of memory without changing the (standard DTW) algorithm or its results is to replace the cost matrix with two vectors, a row and a column vector of cost entries, as the cost of distant previous elements is irrelevant. However, while this reduces the memory footprint, it does not reduce the workload.

Under the assumption that the timing of the two sequences to be aligned does not deviate more than a certain maximum deviation Δt_{max}, neither the distances nor the cost has to be computed for matrix entries outside a band with the width of $2\Delta t_{\mathrm{max}}$ centered around the diagonal from start $(0, 0)$ to stop $(\mathcal{N}_{\mathrm{A}}-1, \mathcal{N}_{\mathrm{B}}-1)$. This optimization has first been proposed by Sakoe and Chiba [1] and is visualized in Figure 10.4b.

If the slope of the alignment path is constrained or, in other words, the tempo difference between the two sequences is limited, then matrix entries outside of a trapezoid spanned by this tempo relationship do not have to be computed. This optimization has first been proposed by Itakura [3] and is visualized in

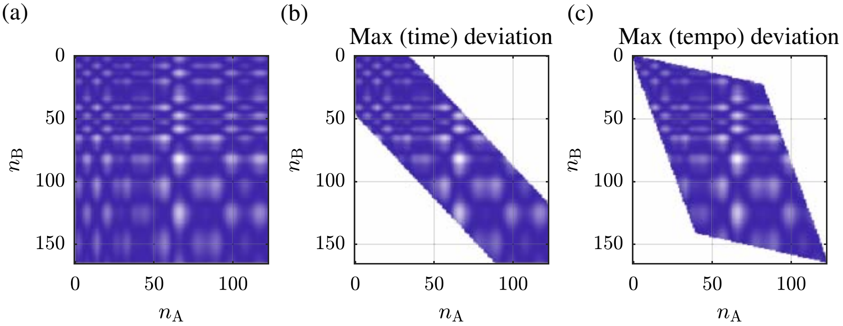


Figure 10.4 Path restrictions for performance optimizations of DTW: original distance matrix (a), matrix with maximum time deviation constraint (b), and matrix with maximum tempo deviation (c).

Figure 10.4c. If start and end points of the two sequences are not necessarily a perfect match, for example, in the case of silent frames at the beginning and end of a sequence, this approach can be combined with the optimization assuming a maximum time deviation (see above) to allow horizontal and vertical movement along the matrix edges at start and end. Otherwise, the path restrictions might become too narrow if the sequences do not start and stop at exactly the same point.

Multiscale DTW is based on the idea of processing the input sequences at different time resolutions. More specifically, the time resolution of the input sequences can be reduced by both using larger block sizes for computing the distances or low-pass filtering and down-sampling the existing sequences. The extracted path through this down-sampled distance matrix can then be used to define a sliding window to be used for a second matrix with finer resolution. This process can be done in two stages [4] or iteratively in multiple stages [2, 5].

The DTW algorithm is not a real-time algorithm, as it requires both complete sequences for processing. It is possible to adapt DTW to a real-time context when the alignment path is only computed within a sliding local window [6], although in order to take advantage of the robustness of DTW, this still requires a high algorithmic latency.

10.2 Audio-to-Audio Alignment

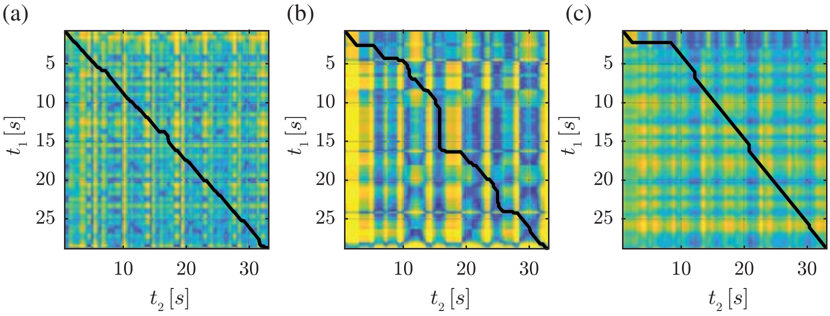
Audio-to-audio alignment describes the process of retrieving corresponding points in time between two audio signals with the same or a similar content. The result maps points in time in one audio signal to points in time in the other audio signal.

The knowledge of those synchronization points enables a variety of different use cases such as

- quick browsing and navigation to certain parts in recordings in order to easily compare parts auditorily [5, 6],
- adjusting the timing of one recording to that of a second by means of a dynamic time stretching algorithm to synchronize, e.g. different voices of a homophonic arrangement or a dubbed version with the original,
- automated synchronization of a (dubbed) studio recording with the original, possibly distorted, recording, and
- musicological analysis of the timing information contained in the alignment path of several performances of the same score in order to compare the tempo and timing to a given reference music performance.

Standard audio-to-audio alignment approaches often use DTW as introduced above. They mainly vary in terms of the features extracted from the audio signal and the subsequent computation of the distance matrix. Hu et al. compute a simple pitch chroma as input feature [7]. The distance is the Euclidean distance between all pitch chroma pairs of the two audio sequences. Turetsky and Ellis use several Short-time Fourier Transform (STFT)-based features such as the power and the first-order difference in time and frequency and then apply the cosine distance as similarity measure [4]. Dixon and Widmer argue that the main objective of audio-to-audio alignment is the synchronization of the onset times and propose to use a spectral flux-based feature subjected to Half Wave Rectification (HWR) [6]. The feature is computed in semitone bands and the Euclidean distance is used as distance measure. In order to achieve pseudo-real-time alignment, they use a modified DTW approach that estimates the optimal local path within a sliding window (see Section 10.1.3). Müller et al. compute the pitch chroma via the energy outputs of a filterbank. They use multiscale DTW as described in Section 10.1.3 to efficiently compute the alignment path [5]. Kirchhoff and Lerch point out that the type of features used for audio-to-audio alignment depends on the similarity to be evaluated between the signals; the distances can be computed, e.g. in the pitch domain, timbre domain, rhythm domain, or envelope domain [8]. They propose to train a two-class linear discriminant analysis (LDA) classifier (*on path* vs. *not on path*) instead of a vector distance; this combines feature weighting and distance computation automatically.

Figure 10.5 visualizes the feature dependency of DTW by showing the path result of a standard DTW implementation for a pair of audio recordings of the same jazz standard. The result for the pitch chroma-based distance matrix on the left is closest to the true alignment path, while the Root Mean Square (RMS) and Mel Frequency Cepstral Coefficient (MFCC)-based distance matrices fail to align the two files correctly in this case.



Repo 1: plotDtwFeatures.m

Figure 10.5 Comparison of different features for computing the distance matrix and its impact on the DTW path result; pitch chroma (a), RMS (b), and Mel Frequency Cepstral Coefficients (MFCCs) (c).

10.3 Audio-to-Score Alignment

Systems for the synchronization of an audio signal with a musical score (frequently in Musical Instrument Digital Interface (MIDI) format) are sometimes categorized with respect to their real-time capabilities. There seems to be some consensus to refer to real-time systems as *score following* systems, and to call nonreal-time (or offline) implementations *audio-to-score alignment* systems.

Possible applications of such alignment systems (see also [9]) include

- linking notation and music performance in applications for musicologists to enable working on a symbolic notation while listening to a real performance,
- using the alignment cost as a distance measure for finding the best-matching document in a database (i.e. retrieve an audio signal for a score query or vice versa),
- the musicological comparison of different performances of the same musical score,
- automatic accompaniment systems,
- the construction of an extended score describing a selected performance by adding information on dynamics, mix information, or lyrics, and
- musical tutoring or coaching systems for which the timing of a recorded performance is evaluated per note.

The core challenge of systems synchronizing audio and score is the proper handling of the different input modalities audio and symbolic score. While the features for audio-to-audio alignment are extracted from audio signals, audio-to-score alignment systems need to compare features extracted from audio with features extracted from the score, leading to potential challenges in computing reliable distance measures.

10.3.1 Real-Time Systems

Historically, the research on matching a predefined score automatically with a music performance can be traced back to 1984. At that time, both Dannenberg and Vercoe independently presented systems for the automatic (computer-based) accompaniment of a monophonic input source in real time [10, 11]. In the following years, these systems were improved for higher accuracy by learning from real-world performances [12] or enhanced by allowing polyphonic input sources, increasing their robustness against musical ornaments, and by using multiple agent systems [13, 14].

Baird et al. proposed a score following system working with MIDI input (from a performance) based on the concept of musical segments as opposed to single musical events; the tracking algorithm itself is not described in detail [15, 16]. Heijink and Desain et al. presented a score following system that takes into account structural information as well. It uses a combination of dynamic programming (DP) and strict pitch matching between performance and score [17, 18].

While many of studies presented above focus on the score following part rather than audio processing itself, Puckette and Lippe worked on systems with audio-only input with monophonic input signals such as clarinet, flute, or vocals [19, 20]. Vantomme proposed a monophonic score following system that uses temporal patterns from the performer as its primary information [21]. From a local tempo estimate, the next event's onset time is predicted and the distance between expected onset time and extracted onset time is evaluated. In the case of an "emergency," pitch information is used as a fall back. Grubb and Dannenberg presented a system following a monophonic vocal performance. It uses the fundamental frequency, spectral features, as well as amplitude changes as features for the tracking process [22, 23]. The estimated score position is calculated based on a probability density function (PDF) conditioned on a distance computed from the previous score event, from the current observation, and from a local tempo estimate. Raphael published several approaches to score following implementing probabilistic modeling and machine learning approaches [24–26]. Cano et al. presented a real-time score following system for monophonic signals based on an Hidden Markov Model (HMM) [27]. They used the features zero crossing rate, energy, and its derivative plus three features computed from the fundamental frequency. Orio et al. introduced a score following system for polyphonic music which utilizes a two-level HMM modeling each event as a state in one level, and modeling the signal with attack, sustain, and rest phase in a lower-level [28, 29]. They used a so-called *Peak Structure Distance (PSD)* that represents the energy sum of band-pass filter outputs with the filters centered around the harmonic series of the pitch of the score event under consideration. Cont proposed a polyphonic score following system using hierarchical Hidden Markov Models HMMs using previously learned pitch templates for multiple fundamental frequency matching [30]. Dorfer et al. introduced Reinforcement Learning into the task of score following and showed that accuracy could be improved with an end-to-end system using images of sheet music [31].

10.3.2 Non-Real-Time Systems

While the publications presented above deal with score following as a real-time application, the systems presented below deal with the closely related topic of non-real-time audio-to-score alignment.

Most approaches use DP techniques for reliable alignment [32]. Orio and Schwarz presented an alignment algorithm for polyphonic music based on DTW which combines several local distances (similarity measures) [33]. It uses the PSD as described above and a *delta of PSD* (ΔPSD) modeling a kind of onset probability; it also uses a silence model for low-energy frames [28]. Meron and Hirose proposed a similar approach with several easy-to-compute audio features and suggested postprocessing of the alignment results to improve the alignment robustness [34]. The system by Arifi et al. attempts to segment the audio signal into (polyphonic) pitches and performs DP to align a MIDI file to the extracted data [35, 36]. Turetsky and Ellis avoided the problems of calculating a direct similarity measure between score and audio by generating an audio file from the (reference) MIDI file and aligning the two audio sequences with audio-to-audio alignment [4]. For the alignment itself, a DP approach is used. Similarly, Dannenberg and Hu generated an audio signal from the MIDI file to align two audio sequences [7, 37]. They calculate the distance measure based on the 12-dimensional pitch chroma. The alignment path is then calculated by a DP approach. Shalev-Shwartz et al. presented a non-real-time system for audio-to-score alignment utilizing DP [38]. Their algorithm features a training stage for the weighting the features for the distance measure. They derived a confidence measure from audio and MIDI similarity data and trained a weight vector for these features to optimize the alignment accuracy over the training set. The audio feature set contains simple pitch-style features extracted by band-pass filtering, derivatives in spectral bands to measure onset probability, and a time deviation from the local tempo estimate. An alignment system of Müller et al. is also based on DP [39]. While targeting piano music, they claim genre independence. For the pitch feature extraction, they used a (zero-phase) filterbank with each band's center frequency located at a pitch on the equal-tempered scale; the filter outputs are used to extract onset times per pitch. Devaney proposed a postprocessing step for fine-alignment of onsets marked as simultaneous in the score by applying a HMM to a previously computed DTW result [40].

In summary, the standard approach to audio-to-score alignment consists of three major processing steps: first, the audio feature extraction which in most cases approximates a pitch-like representation with onset information; timbre and loudness are usually too performance-specific to be used for audio-to-score alignment. Second, a similarity or distance measure between audio and symbolic (score) features is computed. Finally, the actual alignment or path-finding algorithm that is either based on DP, DTW, or on HMMs is applied. To allow for a useful distance matrix as input of a time-warping approach, a meaningful audio-to-score distance has to be computed. On the one hand, the score can be transformed into a more audio-like representation (ultimately by directly rendering the MIDI signal into an audio signal) that allows audio features to be extracted from the score; on the other hand, the audio signal can be converted into a more score-like symbolic format, ultimately by transcribing the signal completely, and a score feature can be extracted from the audio. Alternatively, a distance measure between audio and score can be learned by data-driven methods [41].

10.4 Evaluation

The objective in evaluating alignment systems is the comparison of two sequences of time-stamps and how well they match each other. Therefore, a set of pairs of audio files (or one audio and one time-annotated score file) with clearly defined synchronization points is required as ground truth to evaluate the accuracy of an estimated alignment path. Similar to other tasks, properly annotated data are scarce for the evaluation of both audio-to-audio and audio-to-score alignment. Another important consideration is the time resolution of the ground truth labels. Common variants for audio-to-score alignment include note-level annotations, beat-level annotations, and bar-level annotations. The necessity of higher or lower resolutions is usually defined by the use-case. For score visualization, for example, matching audio at a bar level might be sufficient. For synchronization to trigger accompanying audio events, a much higher-time resolution is necessary. Due to the missing score information, audio-to-audio alignment is generally evaluated against the time-stamps of short blocks after blocking the signals.

Pauses or rests and held notes pose a problem to the evaluation of alignment systems, as there is no ground truth for “correctly” progressing through a rest or a *fermata*. Such pauses generally have to be treated differently or ignored completely in the evaluation. Similarly, long-held notes can become a problem as they do not contain events to align.

10.4.1 Metrics

Cont et al. introduce the following metrics for audio-to-score alignment: (i) the *missed note rate*, i.e. the percentage of missed notes, (ii) the *misalign rate*, i.e. the percentage of notes with a time deviation to the references greater than a tolerance, (iii) the *piece completion*, i.e. the percentage of events until only misaligned events follow toward the end of the piece, (iv) the *average imprecision*, i.e. the average absolute deviation between ground truth and estimated event (compare Eq. 6.6) and (v) the *variance of error*, i.e. the variance of the deviations between ground truth event and corresponding estimated event.

For audio-to-audio alignment, the following metrics have been proposed [8]: (i) the *mean deviation*, i.e. the average offset between the estimated and the ground truth time, (ii) the *mean absolute deviation* similar to the "average imprecision" mentioned above, (iii) the *maximum deviation*, i.e. the largest deviation per file to have a worst-case estimate, and (iv) the *relative number of matching path points* between estimated times and reference times.

10.4.2 Data

There exist several ways of generating a ground truth data set for audio-to-audio alignment systems. Corresponding points in time can be (manually or semiautomatically) annotated, onset times can be monitored and stored during a recording of a computer-monitored instrument, or MIDI files can be rendered to audio by means of a sample player.

The manual annotation of (alignment) points in time is a rather arduous process and thus generally only a few points per test file are labeled (see also Section 9.3.3.2). However, Gadermaier and Widmer argue that modern alignment systems are sufficiently reliable if only one of multiple performances of one piece is being annotated carefully while the others can be annotated automatically [42]. However, automatic annotation of audio data followed by manual verification of the labels might lead to an unfair evaluation advantage of the system used to create the labels compared to others [43]. Still, it could be a valid option to measure algorithmic improvements.

The MAESTRO dataset, providing note-level onset times, is a good resource for evaluating piano music alignment [44]. The drawback of using piano-generated data is its restriction to solo piano music.

Audio can also be synthesized with variations in timing and timbre through varying tempo in a MIDI editor and synthesizing audio through different samplers. The disadvantage of using synthesized samples is that their properties might differ from "real-world" signals in terms of sound quality and signal complexity.

Alternatively, it is possible to artificially generate modified pairs of audio signals. In the audio domain, this can be done with a dynamic time-stretching algorithm, i.e. an audio-processing algorithm able to change the tempo without changing the pitch of an audio signal [45]. This allows for high accuracy of the data set while allowing to test with a wide range of musical styles and instrumentations. In this case, however, the validity of the data depends (i) on the realistic dynamic use of stretch factors and (ii) on the audio quality of the stretching engine. Furthermore, as the test data originate from the same audio signal, the algorithm's performance might be overestimated.

10.5 Exercises

10.5.1 Questions

1. Given two time series v_1 of length 4 and v_2 of length 5, what are the minimum and maximum possible DTW alignment path lengths?
2. Given two time series

$$v_1 = [0\ 2\ 3\ 1]$$
$$v_2 = [0\ 1\ 2\ 0\ 1],$$

   (a) compute the distance matrix $\boldsymbol{D}_{\mathrm{AB}}$ using the absolute difference,
   (b) compute the cost matrix $\boldsymbol{C}_{\mathrm{AB}}$, and
   (c) compute the correct path indices with DTW.

10.5.2 Assignments

1. Distance Matrix
   (a) Implement a function `[D] = compute_dist(v_1,v_2, cDistanceMeasure)` which takes two series of feature vectors `v_1` and `v_2` and computes the pairwise distance between them. The output `D` is nonnegative and its dimensions correspond to the number of observations in series `1` times the number of observations in sequence `2`. `cDistanceMeasure` can be
      - `'euclidean'`: Euclidean distance,
      - `'manhattan'`: Manhattan distance, or
      - `'cosine'`: Cosine distance.
2. Dynamic Time Warping
   (a) Implement a function `[p, C] = compute_dtw(D)` which takes the distance matrix as input argument and finds the optimal path through this matrix. It returns the path with all the path points in matrix indices (dimensions 2 × path length) and the computed cost matrix `C` of the same dimensions as `D`.
   (b) Test your functions with the example from the chapter.
   (c) Take two performances of the same piece from the MAESTRO dataset.[2] Compute the pitch chroma of both with the function `extract_pitch_chroma` from Section 7.7.2, then compute distance matrix and DTW for

---

[2] MAESTRO. https://magenta.tensorflow.org/datasets/maestro. Last retrieved on 27 November 2021.

the pair. Discuss the tempo differences between the two performance as indicated by your path. Note: if you have an older machine, focus on shorter recordings. If you haven't implemented `extract_pitch_chroma`, use the following instructions:

- Implement a function `[fs, x] = read_audio(cPath2Audio)` which reads a wave audio file with scipy.io.wavfile.read and returns the sample rate `fs` and a numpy array `x` with dimensions `numSamples`× `numChannels`. Ensure that the maximum possible value range of `x` is always $[-1, 1]$ regardless of the wordlength of the input file. Support the following wordlengths/input types: `uint8`, `int16`, `int32`, `float32`
- Implement a function `[x_m] = downmix_audio(x)` which takes a numpy array `x` with dimensions `numSamples` × `numChannels` and returns the downmixed numpy array `x_m` with dimensions `numSamples` × 1.
- Implement a function `[x_B, t_B] = block_audio(x_m, blockSize, hopSize, fs)` which returns a numpy array `x_B` (dimension `N` × `blockSize`) and a vector t_B (dimension `N` × 1) for blocking the input audio signal into overlapping blocks. Make sure that each sample of the audio signal is included in at least one of the `N` blocks. `t_B` will refer to the start time of each block in seconds.
- Implement a function `[w] = compute_hann(windowLength)` that computes a Hann window of length `windowLength` and returns the window in numpy vector `w`. Before you implement, investigate the difference between choosing a "symmetric" vs. a "periodic" window function and choose the "periodic" implementation.
- Implement a function `[X, fInHz, t_B] = compute_spectrogram(x_B, t_B, fs)` that computes the magnitude spectrum for each block of audio in `x_B` and returns the magnitude spectrogram `X` (dimensions `blockSize/2+1` × `N`), a frequency vector `fInHz` (dimensions `blockSize/2+1` × 1) containing the central frequency of each bin, and the previously computed time stamp vector `t_B`.
- Write a function `[v_pc] = extract_pitch_chroma(X, fs, tfInHz = 440)` which returns the pitch chroma array `v_pc` (dimensions 12 × numBlocks). `X` is the magnitude spectrogram, `fs` is the sample rate, and `tfInHz` is the tuning frequency. Compute the pitch chroma for the 3 octave range C3–B5. Make sure your implementation works with different sample rates and block sizes. You will need to adjust the semitone bands based on the input parameter `tfInHz`. Normalize each individual chroma vector to a length of 1 (L2 norm).

References

1 Hiroaki Sakoe and Seibi Chiba. Dynamic Programming Algorithm Optimization for Spoken Word Recognition. *Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, 1978.

2 Meinard Müller. *Information Retrieval for Music and Motion*. Springer-Verlag, Berlin, 2007. ISBN 978-3-540-74047-6.

3 Fumitada Itakura. Minimum Prediction Residual Principle applied to Speech Recognition. *Transactions on Acoustics, Speech, and Signal Processing*, 23(1):67–72, February 1975. ISSN 0096-3518. doi: 10.1109/TASSP.1975.1162641. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1162641.

4 Robert J Turetsky and Daniel P W Ellis. Ground-Truth Transcriptions of Real Music from Force-Aligned MIDI Syntheses. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Baltimore, MD, 2003.

5 Meinard Müller, Henning Mattes, and Frank Kurth. An Efficient Multiscale Approach to Audio Synchronization. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Victoria, 2006.

6 Simon Dixon and Gerhard Widmer. MATCH: A Music Alignment Tool Chest. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, London, September 2005.

7 Ning Hu, Roger B Dannenberg, and George Tzanetakis. Polyphonic Audio Matching and Alignment for Music Retrieval. In *Proceedings of the Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, New Paltz, 2003. IEEE.

8 Holger Kirchhoff and Alexander Lerch. Evaluation of Features for Audio-to-Audio Alignment. *Journal of New Music Research*, 40(1):27–41, 2011. doi: 10.1080/09298215.2010.529917. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2015/04/Kirchhoff_Lerch_2011_Evaluation.

9 Ferréol Soulez, Xavier Rodet, and Diemo Schwarz. Improving Polyphonic and Poly-Instrumental Music to Score Alignment. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Baltimore, MD, 2003.

10 Roger B Dannenberg. An On-Line Algorithm for Real-Time Accompaniment. In *Proceedings of the International Computer Music Conference (ICMC)*, Paris, 1984.

11 Barry L Vercoe. The Synthetic Performer in the Context of Live Performance. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 199–200, Paris, 1984.

12 Barry L Vercoe and Miller S Puckette. Synthetic Rehearsal: Training the Synthetic Performer. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 275–278, Vancouver, 1985.

13 Joshua J Bloch and Roger B Dannenberg. Real-Time Computer Accompaniment of Keyboard Performances. In *Proceedings of the International Computer Music Conference (ICMC)*, Vancouver, 1985.

14 Roger B Dannenberg and Hirofumi Mukaino. New Techniques for Enhanced Quality of Computer Accompaniment. In *Proceedings of the International Computer Music Conference (ICMC)*, Cologne, September 1988.

15 Bridget Baird, Donald Blevins, and Noel Zahler. The Artificially Intelligent Computer Performer: The Second Generation. *Interface – Journal of New Music Research*, 19:197–204, 1990.

16 Bridget Baird, Donald Blevins, and Noel Zahler. Artificial Intelligence and Music: Implementing an Interactive Computer Performer. *Computer Music Journal*, 17(2):73–79, 1993.

17 Hank Heijink. *Matching Scores and Performances*. Master's thesis, Nijmegen University, 1996.

18 Peter Desain, Henkjan Honing, and Hank Heijink. Robust Score-Performance Matching: Taking Advantage of Structural Information. In *Proceedings of the International Computer Music Conference (ICMC)*. International Computer Music Association, Thessaloniki, September 1997.

19 Miller S Puckette and Cort Lippe. Score Following in Practice. In *Proceedings of the International Computer Music Conference (ICMC)*, San Francisco, CA, 1992.

20 Miller S Puckette. Score Following Using the Sung Voice. In *Proceedings of the International Computer Music Conference (ICMC)*, Banff, 1995.

21 Jason D Vantomme. Score Following by Temporal Pattern. *Computer Music Journal*, 19(3):50–59, 1995. 00023.

22 Lorin Grubb and Roger B Dannenberg. A Stochastic Method of Tracking a Vocal Performer. In *Proceedings of the International Computer Music Conference (ICMC)*, Thessaloniki, September 1997.

23 Lorin Grubb and Roger B Dannenberg. Enhanced Vocal Performance Tracking Using Multiple Information Sources. In *Proceedings of the International Computer Music Conference (ICMC)*, Ann Arbor, MI, 1998.

24 Christopher Raphael. Automatic Segmentation of Acoustic Musical Signals Using Hidden Markov Models. *Transactions on Pattern Analysis and Machine Intelligence*, 21(4):360–370, 1999.

25 Christopher Raphael. A Probabilistic Expert System for Automatic Musical Accompaniment. *Journal of Computational and Graphical Statistics*, 10(3):487–512, 2001.

26 Christopher Raphael. A Hybrid Graphical Model for Aligning Polyphonic Audio with Musical Scores. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Barcelona, 2004.

27 Pedro Cano, Alex Loscos, and Jordi Bonada. Score-Performance Matching using HMMs. In *Proceedings of the International Computer Music Conference (ICMC)*, Beijing, China, 1999.

28 Nicola Orio and Francois Déchelle. Score Following Using Spectral Analysis and Hidden Markov Models. In *Proceedings of the International Computer Music Conference (ICMC)*, Habana, September 2001.

29 Nicola Orio, Serge Lemouton, and Diemo Schwarz. Score Following: State of the Art and New Developments. In *Proceedings of the Conference of New Interfaces for Musical Expression (NIME)*, Montreal, 2003.

30 Arshia Cont. Realtime Audio to Score Alignment for Polyphonic Music Instruments using Sparse Non-Negative Constraints and Hierarchical HMMs. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 5, Toulouse, 2006.

31 Matthias Dorfer, Jan Haji? Jr, Andreas Arzt, Harald Frostel, and Gerhard Widmer. Learning Audio–Sheet Music Correspondences for Cross-Modal Retrieval and Piece Identification. *Transactions of the International Society for Music Information Retrieval*, 1(1):22–33, September 2018. ISSN 2514-3298. doi: 10.5334/tismir.12. URL http://transactions.ismir.net/articles/10.5334/tismir.12/. Number: 1 Publisher: Ubiquity Press.

32 Edward Wilson Large. Dynamic Programming for the Analysis of Serial Behaviors. *Behavior Research Methods, Instruments, and Computers*, 25(2):238–241, 1993.

33 Nicola Orio and Diemo Schwarz. Alignment of Monophonic and Polyphonic Music to a Score. In *Proceedings of the International Computer Music Conference (ICMC)*, Habana, September 2001.

34 Yoram Meron and Keikichi Hirose. Automatic Alignment of a Musical Score to Performed Music. *Acoustical Science & Technology*, 22(3):189–198, 2001.

35 Vlora Arifi. *Algorithmen zur Synchronisation von Musikdateien im Partitur-, MIDI- und PCM-Format*. doctoral thesis, Rheinische Friedrich-Wilhelms-Universität, Bonn, 2002.

36 Vlora Arifi, Michael Clausen, Frank Kurth, and Meinard Müller. Score-PCM Music Synchronization based on Extracted Score Parameters. In *Proceedings of the 2nd International Symposium on Computer Music Modeling and Retrieval (CMMR)*, Trondheim, 2004.

37 Roger B Dannenberg and Ning Hu. Polyphonic Audio Matching for Score Following and Intelligent Audio Editors. In *Proceedings of the International Computer Music Conference (ICMC)*, Singapore, 2003.

38 Shai Shalev-Shwartz, Joseph Keshet, and Yoram Singer. Learning to Align Polyphonic Music. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Barcelona, 2004.

39 Meinard Müller, Frank Kurth, and Tido Röder. Towards an Efficient Algorithm for Automatic Score-to-Audio Synchronization. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Barcelona, 2004.

40 Johanna Devaney. Estimating Onset and Offset Asynchronies in Polyphonic Score-Audio Alignment. *Journal of New Music Research*, 43(3):266–275, July 2014. ISSN 0929-8215. doi: 10.1080/09298215.2014.890630. URL http://dx.doi.org/10.1080/09298215.2014.890630.

41 Cyril Joder, Slim Essid, and Gaël Richard. Learning Optimal Features for Polyphonic Audio-to-Score Alignment. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(10):2118–2128, October 2013. ISSN 1558-7924. doi: 10.1109/TASL.2013.2266794. Conference Name: IEEE Transactions on Audio, Speech, and Language Processing.

42 Thassilo Gadermaier and Gerhard Widmer. A Study of Annotation and Alignment Accuracy for Performance Comparison in Complex Orchestral Music. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Delft, Netherlands, October 2019. URL http://arxiv.org/abs/1910.07394.

43 Arshia Cont, Diemo Schwarz, Norbert Schnell, and Christopher Raphael. Evaluation of Real-Time Audio-To-Score Alignment. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Vienna, 2007.

44 Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset. In *Proceedings of the International Conference on Learning Representations (ICLR)*, New Orleans, LA, January 2019. URL http://arxiv.org/abs/1810.12247.

45 Gianpaolo Evangelista. Time and Frequency Warping Musical Signals. In Udo Zölzer, editor, *DAFX – Digital Audio Effects*, pages 439–463. John Wiley & Sons Ltd, 2002. ISBN 0-471-49078-4.

Part III

Music Identification, Classification, and Assessment

11

Audio Fingerprinting

Fingerprinting aims at identifying audio recordings in a previously assembled database. More specifically, each recording is represented by a so-called fingerprint, a unique and compact digest summarizing the (perceptually) relevant aspects of the recording. The fingerprint of an unknown recording is then the query to identify this recording in a database containing previously extracted fingerprints and to retrieve associated metadata. In contrast to many other tasks presented in the book, fingerprinting does *not* attempt to extract musical properties from the audio signal but targets identifying a specific recording (as opposed to a specific song). Different music performances (or recordings) of the same song should therefore have different fingerprints. However, a recording still should be correctly identified even when subjected to quality degradation such as perceptual audio coding, added noise, distortions, and other common signal manipulations.

There are two main areas of application: (i) broadcast monitoring to determine which songs have been played and how often, to allow rights holders to estimate royalties and track possible license violations, and (ii) end consumer services which enable the user to easily identify audio in order to provide metadata such as artist name, song title, album art, or other added-value services. A detailed overview of other applications for fingerprinting has been given by Cano et al. [1].

Fingerprinting is not to be confused with *Watermarking*. Although both are used for identifying recordings, watermarking embeds a perceptually unnoticeable data block directly in the audio data, utilizing methods similar to perceptual audio coding while fingerprinting does not modify the audio signal. Watermarking thus enables the content provider to embed different watermarks in the same audio content. It allows, for example, to embed a user-specific watermark in a specific copy of the recording in order to identify this specific user copy of the recording later. This is not possible with fingerprinting, which identifies a recording by its audio content itself, not by its embedded data. As watermarking can embed metadata directly it can also give the user direct access to these additional data (e.g. song

An Introduction to Audio Content Analysis: Music Information Retrieval Tasks & Applications,
Second Edition. Alexander Lerch.
© 2023 The Institute of Electrical and Electronics Engineers, Inc. Published 2023 by John Wiley & Sons, Inc.

Table 11.1 Main properties of fingerprinting and watermarking in comparison.

Property	Fingerprinting	Watermarking
Allows legacy content indexing	+	−
Allows embedded data	−	+
Leaves signal unchanged	+	−
Identification of	Recording	User or interaction

title or artist name) without the need for a database connection. The major advantage of fingerprinting over watermarking is that the audio signal does not have to be modified. Therefore, fingerprinting can cover legacy audio recordings distributed previously or through other distribution channels. Furthermore, it avoids the risk that audio signal modifications can possibly degrade the audio quality (with similar quality degradation as caused by perceptual encoders). Table 11.1 compares the main properties of fingerprinting and watermarking.

A fingerprinting system consists of two basic building blocks, the fingerprint extraction and a database of previously extracted fingerprints coupled with unique identifiers or additional metadata about the piece of music. Figure 11.1 visualizes these blocks; the upper part of the graph shows the process of adding new entries to the database (done by the service provider) and the lower part shows the query for a recording by a client.

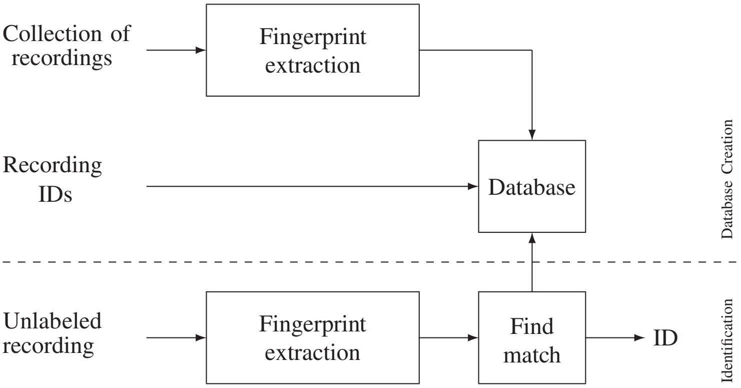

Collection of recordings
Fingerprint extraction
Recording IDs
Database
Database Creation
Unlabeled recording
Fingerprint extraction
Find match
ID
Identification

Figure 11.1 General framework for audio fingerprinting. The upper part visualizes the training phase and the lower part the query phase.

The requirements on a general fingerprinting system have been summarized by Cano et al. [2] as:

- *Accuracy and reliability*: high number of correct identifications, i.e. True Positives (TPs), compared to the number of missed identifications, i.e. False Negatives (FNs), and wrong identifications, i.e. False Positives (FPs).
- *Robustness and security*: high accuracy even in the case of a heavily distorted signal. Possible distortions include lossy compression, added noise, equalization, interference, and nonlinearities of the transmission path. Sophisticated systems should also be robust against changes in tempo and pitch.
- *Granularity*: the shorter the length of an excerpt required for its identification, the user-friendlier (most systems require at least several seconds).
- *Versatility*: independence of detection from the file format and the file origin as well as an application-independent implementation.
- *Scalability*: good performance on very large databases with a large number of simultaneous identification queries.
- *Complexity*: low computational cost of both extracting a fingerprint and finding this fingerprint in the database.

11.1 Fingerprint Extraction

The fingerprint should be a unique identifier of a recording that allows unambiguous identification of a recording while being robust against quality degradations from, e.g. equalization, bandwidth restriction, audio coding, or background noise. For these reasons, it is sometimes referred to as a *perceptual hash*. At the same time, it should be as compact as possible.

Since the fingerprint should be robust against bandwidth restrictions and audio format, the two most common preprocessing steps are down-mixing to a single channel and down-sampling to a lower sample rate (usually 5–20 kHz). Applying a high-pass filter discards frequency components below the lowest transmittable frequency of phones in an optional preprocessing step.

A system that can be seen as an early predecessor of today's fingerprinting systems targeted the detection of advertisements in broadcast streams. Lourens used the advertisement's energy envelope and selected a "unique" section serving as fingerprint [3]. In most contemporary systems, the features are extracted in the frequency domain. These features include Mel Frequency Cepstral Coefficients (MFCCs) [4–6], a spectral flatness measure, and a spectral crest factor per frequency band [7], a spectral centroid per subband [8], band energies [9] or (the sign of) energy band differences [10], carefully selected spectral peaks [11, 12], statistical moments of subbands [13], and modulation frequency features [14].

Frequently, the feature extraction process yields multiple values per processing block (often represented with a word length of 32 Bits), leading to too much data to serve as the fingerprint directly. In order to compress the information and to decrease the memory footprint, many of these features (or the feature derivatives) are quantized into a binary or ternary representation. There have also been attempts at learning a fingerprint extraction algorithm automatically from the audio [15].

The resulting fingerprint then contains a unique series of quantized feature values or feature vectors.

11.2 Fingerprint Matching

The extracted fingerprint, representing the unknown recording, has to be compared against all previously stored fingerprints in the database. Often, the extracted fingerprint only consists of a segment of the song; after all, a user does not necessarily want to wait to record the whole song before identifying it. The database, however, has to contain all potential fingerprints of the whole recording to ensure a recording can be identified regardless of the position in the recording. As the database can be large, the similarity (or distance) measure has to be computationally efficient. Common measures include a correlation measure [16, 17], the Euclidean distance [8, 9, 18], and the Manhattan distance (which in the case of binary input equals the *Hamming distance*) [10, 19], but there exist many possible alternatives (see, e.g. [4]).

Even with a fast-to-compute similarity measure, it might not be possible to compare every query fingerprint against all stored fingerprints in a large database due to workload and response time constraints. Various ways for performance improvement have been proposed. One simple way would be to presort the database entries by their “popularity” in order to reduce search time for songs with frequent queries. It is also possible to precompute distances between the stored fingerprints in order to find different entry points for the query [9] or to use different similarity measures, an efficient one to discard many database entries in a first run and a second more accurate similarity measure to be computed on a selected small subset [20].

As the extracted fingerprint is unlikely to match the stored fingerprint perfectly, it has to be considered carefully what distance value constitutes a match. It depends, as Cano et al. note, on the used fingerprint model, the discriminative information of the query, the similarity of the fingerprints in the database, and the database size [21]. This includes not only optimizing the number of correct detections of songs in the database but also correctly reporting if songs are not in the database.

11.3 Fingerprinting System: Example

To allow for a better understanding of the process of audio fingerprinting, a widespread and frequently referenced system will be explained in detail in the following: the Philips fingerprinting system as published by Haitsma and Kalker [10], in which one fingerprint comprises of 256 subsequent subfingerprints.

After the signal is down-mixed to one channel and down-sampled to a sample rate of 5 kHz, it is subjected to a (von-Hann-windowed) short-time fourier transform (STFT). The block length is 0.37 s and the hop size is 11.6 ms. The large block overlap ratio increases the system's robustness against time-shift operations.

Figure 11.2 shows an overview of the subfingerprint extraction for one block of audio. The magnitude spectrum is divided into 33 nonoverlapping bands in the range 300–2000 Hz. The frequency bandwidth increases logarithmically to take into account the nonlinear frequency resolution of the human ear (see Section 7.1).

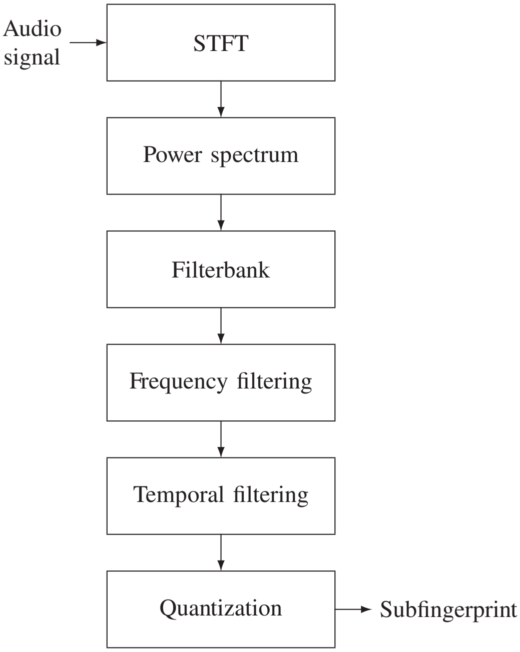


Figure 11.2 Flowchart of the extraction process of subfingerprints in the Philips system.

The energy E per band k is then used to derive a binary result by using both the time and frequency derivative:

$$v_{\mathrm{FP}}(k,n) = \begin{cases} 1 & \text{if } (\Delta E(k,n) - \Delta E(k,n-1)) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{11.1}$$

with

$$\Delta E(k,n) = E(k,n) - E(k+1,n). \tag{11.2}$$

This results in a 32-Bit word per STFT; Haitsma et al. refer to this word as *subfingerprint*. One complete fingerprint consists of 256 subsequent subfingerprints and thus spans a time of approximately three seconds. It has a size of $256 \cdot 4$ Byte = 1 kByte. Two example fingerprints are shown in Figure 11.3, one of the original file (a) and one of the same file but encoded and decoded with MPEG–1 Layer 3 (MP3) at a low bit rate (b). Each row is one subfingerprint. The differences between the two fingerprints are shown on the right. Example code for the extraction of this fingerprint is presented in Listing 11.1.

```
% power spectrum
X = abs(X) * 2 / iBlockLength;
X([1 end], :) = X([1 end], :) / sqrt(2); % normalization
X = abs(X).^2;

% group spectral bins in bands
E = H * X;

% extract fingerprint through diff (both time and freq)
SubFingerprint = diff(diff(E, 1, 1), 1, 2);
tf = tf(1:end-1) + iHopLength / (2 * target_fs);

% quantize fingerprint
SubFingerprint(SubFingerprint<0) = 0;
SubFingerprint(SubFingerprint>0) = 1;
```

Repo 2: computeFingerprint.m
Repo 3: computeFingerprint.py

Listing 11.1 Code example for fingerprint extraction.

The distance measure for the database search used by the Philips system is the Manhattan distance; since the fingerprints are binary, the Manhattan distance equals the *Hamming distance*. Haitsma et al. report that a fingerprint length of three seconds is sufficient for the identification of a song from the database.

The database has to contain the series of all subfingerprints of each complete recording. Thus, if the database contains one million songs of approximately five minutes length, it holds more than 25 billion subfingerprints. Even in the case of a highly compressed subfingerprint format and the use of the computationally efficient Hamming distance, this amount of subfingerprints rules out the brute force approach of searching the whole database for each query. Haitsma et al. suggest two methods to improve computational efficiency of the database search, a simple and a more refined method. First, a lookup table is added to the database. This table contains all possible 32-bit subfingerprints which leads to a maximum number of 2^{32} table entries. Each table entry points to a list of occurrences in the database. The lookup table can also be replaced by a hash table for efficiency.

The first proposed method is based on the assumption that at least one of the 256 extracted subfingerprints has an exact match at the correct position in the database. Therefore, only the database entries listed under one of the 256 subfingerprints of the current query have to be evaluated as possible matches. While this approach reduces the workload dramatically, the assumption that there is at least one subfingerprint without a bit error is not necessarily true for audio with more than minor degradations. A large number of bit errors can be expected for highly distorted signals, which means that the correct database entry might never be checked for a match. Even in the relatively high-quality fingerprint shown in Figure 11.3, less than 70 subfingerprints are error-free, all others contain one or more errors with a maximum of 10 errors for one subfingerprint.

That indicates that taking into account possible bit errors might be a prudent approach. However, considering bit errors in the subfingerprints has the disadvantage of increasing the workload again: if *one* bit error is expected per subfingerprint, the number of database queries and thus the computational workload might increase by as much as a factor of 33.

In order to reduce this additional workload while still taking into account possible bit errors, the concept of the *reliability* of a bit error is introduced in the second proposed method. Since the bits of a subfingerprint are computed by energy differences, the likelihood of a bit being flipped (a bit error) is high for small energy differences and low for large differences. Thus, the bits can be ranked by their reliability and only the unreliable bits have to be flipped for the database search.

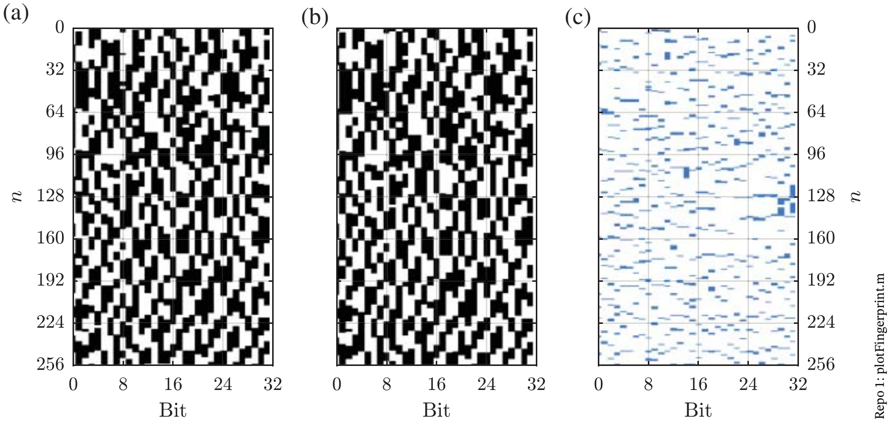


Figure 11.3 Two fingerprints and their difference. (a) Original, (b) mp3 encoded, (c) bit differences between the two fingerprints.

11.4 Evaluation

In contrast to other tasks presented in this book, the evaluation of audio fingerprinting algorithms does not require musical annotations. Therefore, the size of the available datasets is not restricted by a time-consuming manual annotation step. However, the performance of a fingerprinting system is dramatically impacted by the scale of the dataset used for evaluation; obviously, it is easier to identify a song correctly from a set of 1000 recordings than from a set of 1 000 000 recordings. A dataset of that scale and with complete songs, however, is not publicly available.

Given these reasons, the evaluation section of existing audio fingerprinting studies sometimes lacks the rigor of studies on other tasks, impacted by the lack of benchmark results and generally the availability of a dataset. Often, evaluations are carried out on small proprietary datasets of thousands of songs with limited replicability and validity.

Since most audio fingerprinting systems are required to work in scenarios with quality-degraded and distorted audio, experiments with time-shifted audio, added noise, different bandwidths, dynamics processing, and other linear or nonlinear distortions can verify the robustness of the tested algorithm. Robustness against attacks such as time-stretching and pitch-shifting can also be an important consideration if the algorithm is intended for detecting copyright infringements.

The most common metrics are the True Positive Rate (TPR) (also referred to as hit rate) and the False Positive Rate (FPR). These metrics have been introduced in Section 6.1.1. Note that some studies compute the hit rate with a margin: if the correct song is ranked in the list of top N predictions, it is considered a correct detection. Another interesting aspect to evaluate is conveyed by the False Negative Rate (FNR) (also referred to as false rejection rate). This is the ratio of negative responses for recordings that are in the database.

References

1 Pedro Cano, Eloi Batlle, Emilia Gomez, Leandro De C T Gomes, and Madeleine Bonnet. Audio Fingerprinting: Concepts and Applications. In Saman K Halgamuge, and Lipo Wang, editors, *Computational Intelligence for Modelling and Prediction*, pages 233–245. Springer-Verlag, Berlin, 2005.

2 Pedro Cano, Eloi Batlle, Ton Kalker, and Jaap Haitsma. A Review of Audio Fingerprinting. *The Journal of VLSI Signal Processing-Systems for Signal, Image, and Video Technology*, 41(3):271–284, November 2005. ISSN 0922-5773. doi: 10.1007/s11265-005-4151-3. URL http://www.springerlink.com/index/10.1007/s11265-005-4151-3.

3 J G Lourens. Detection and Logging Advertisements Using its Sound. *Transactions on Broadcasting*, 36(3):231–233, 1990. ISSN 00189316. doi: 10.1109/11.59850. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=59850.

4 Eloi Batlle, Jaume Masip, and Enric Guaus. Automatic Song Identification in Noisy Broadcast Audio. In *Proceedings of the International Conference on Signal and Image Processing (SIP)*, Kauai, 2002.

5 Pedro Cano, Eloi Batlle, Harald Mayer, and Helmut Neuschmied. Robust Sound Modeling for Song Detection in Broadcast Audio. In *Proceedings of the Audio Engineering Society Convention*, Munich, 2002. AES.

6 Arunan Ramalingam and Sridhar Krishnan. Gaussian Mixture Modeling of Short-Time Fourier Transform Features for Audio Fingerprinting. *Transactions on Information Forensics and Security*, 1(4):457–463, December 2006. ISSN 1556-6013. doi: 10.1109/TIFS.2006.885036. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4014108.

7 Oliver Hellmuth, Eric Allamanche, Jürgen Herre, Thorsten Kastner, Markus Cremer, and Wolfgang Hirsch. Advanced Audio Identification Using MPEG-7 Content Description. In *Proceedings of the 111th Audio Engineering Society Convention (Preprint No. 5463)*, New York, 2001.

8 Jin S Seo, Minho Jin, Sunil Lee, Dalwon Jang, Seungjae Lee, and Chang D Yoo. Audio Fingerprinting Based on Normalized Spectral Subband Centroids. *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 213–216, 2005. doi: 10.1109/ICASSP.2005.1415684. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1415684.

9 Akisato Kimura, Kunio Kashino, Takayuki Kurozumi, and Hiroshi Murase. Very Quick Audio Searching: Introducing Global Pruning to the Time-Series Active Search. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 3, pages 1429–1432. IEEE, 2001. ISBN 0-7803-7041-4. doi: 10.1109/ICASSP.2001.941198. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=941198.

10 Jaap Haitsma and Ton Kalker. A Highly Robust Audio Fingerprinting System. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Paris, 2002.

11 Cheng Yang. MACS: Music Audio Characteristic Sequence Indexing for Similarity Retrieval. In *Proceedings of the Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 123–126, New Paltz, NY, 2001. IEEE. ISBN 0-7803-7126-7. doi: 10.1109/ASPAA.2001.969558. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=969558.

12 Avery Wang. An Industrial Strength Audio Search Algorithm. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Washington, DC, 2003. URL http://www.cs.northwestern.edu/ pardo/courses/-casa/papers/An Industrial-Strength Audio Search Algorithm.pdf.

13 Sungwoong Kim and Chang D Yoo. Boosted Binary Audio Fingerprint Based on Spectral Subband Moments. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 1, pages I–241–I–244. IEEE, April 2007. ISBN 1-4244-0727-3. doi: 10.1109/ICASSP.2007.366661. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4217061.

14 Somsak Sukittanon and Les E Atlas. Modulation Frequency Features for Audio Fingerprinting. In *Proceedings of the International Conference on Acoustics Speech and Signal Processing (ICASSP)*, page II. IEEE, 2002. ISBN 0-7803-0946-4. doi: 10.1109/ICASSP.2002.1006107. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1006107.

15 Sungkyun Chang, Donmoon Lee, Jeongsoo Park, Hyungui Lim, Kyogu Lee, Karam Ko, and Yoonchang Han. Neural Audio Fingerprint for High-specific Audio Retrieval based on Contrastive Learning. In *Proceedings of the International Conference on Acoustics Speech and Signal Processing (ICASSP)*, February 2021. URL http://arxiv.org/abs/2010.11910.

16 Daniel Schonberg and Darko Kirovski. Fingerprinting and Forensic Analysis of Multimedia. In *Proceedings of the 12th ACM Multimedia Conference*, New York, 2004.

17 José Ignacio Martínez, Jaime Vitola, Adriana Sanabria, and Cesar Pedraza. Fast Parallel Audio Fingerprinting Implementation in Reconfigurable Hardware and GPUs. In *Proceedings of the Southern Conference on Programmable Logic (SPL)*, pages 245–250. IEEE, April 2011. ISBN 978-1-4244-8847-6. doi: 10.1109/SPL.2011.5782656. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5782656.

18 Christopher J C Burges, John C Platt, and Soumya Jana. Extracting Noise-Robust Features from Audio Data. In *Proceedings of the International Conference on Acoustics Speech and Signal Processing (ICASSP)*. IEEE, 2002. ISBN 0-7803-0946-4. doi: 10.1109/ICASSP.2002.1005916. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1005916.

19 Gábor Richly, László Varga, Ferenc Kovacs, and Gábor Hosszu. Short-term Sound Stream Characterization for Reliable, Real-Time Occurrence Monitoring of Given Sound-Prints. In *Proceedings of the 10th Mediterranean Electrotechnical Conference. Information Technology and Electrotechnology for the Mediterranean Countries (CMeleCon)*, volume 2, pages 526–528. IEEE, 2000. ISBN 0-7803-6290-X. doi: 10.1109/MELCON.2000.879986. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=879986. 00011.

20 Thorsten Kastner, Eric Allamanche, Jürgen Herre, and Oliver Hellmuth. MPEG-7 Scalable Robust Audio Fingerprinting. *Proceedings of the 112th Audio Engineering Society Convention*, 2002.

21 Pedro Cano, E Batle, T Kalker, and Jaap Haitsma. A Review of Algorithms for Audio Fingerprinting. In *Proceedings of the Workshop on Multimedia Signal Processing (MMSP)*, pages 169–173. IEEE, 2002. ISBN 0-7803-7713-3. doi: 10.1109/MMSP.2002.1203274. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1203274.

12

Music Similarity Detection and Music Genre Classification

The tasks of detecting music similarity and musical genre have a long history of considerable research interest. These two tasks are combined in one chapter because musical genre can be seen as a category label for grouping musically similar songs and can thus be considered a special case of music similarity.

A system able to predict the similarity between to music recordings or to classify the musical genre of a song allows the retrieval of similar musical pieces and enables applications such as the intuitive browsing of large catalogs of music or automatic playlist generation and music recommendation. An audio-based measure of music similarity is, therefore, of considerable (commercial) interest and has been proposed for decades [1]. There are indications, however, that – at least at the current state of the art – content-based similarity measures for, e.g. recommendation, are consistently outperformed by systems building on contextual metadata [2].

12.1 Music Similarity Detection

While a music genre can be seen as similarity cluster formed by acoustic and musical properties, one might attempt to derive a similarity measure from inter-genre distances. Despite the fact that such inter-genre distances cannot easily be quantified, such a reduction of music similarity to genre similarity is too simplifying (despite the complex nature of defining genre itself). While the task of describing music similarity is at least as subjective as assigning genre labels to music, genre similarity is only one facet of a large variety of factors influencing the perception of music similarity. The understanding of what music is considered similar often depends on the point of view. Two songs might be seen as similar because they are of the same musical genre when they are compared to songs of different genres, but the same songs might be considered completely dissimilar when compared to a cover version of one song. A Disk Jockey (DJ) might want to look for songs with

An Introduction to Audio Content Analysis: Music Information Retrieval Tasks & Applications,
Second Edition. Alexander Lerch.
© 2023 The Institute of Electrical and Electronics Engineers, Inc. Published 2023 by John Wiley & Sons, Inc.

similar tempo or key in her database, but might appreciate if they are different in other aspects. Hence, the definition of music similarity depends on both the individual user and the task at hand.

To give a better impression of the multifaceted and multidimensional nature of music similarity, a few aspects of similarity are listed here with some bibliographical entry points: rhythmic similarity [3, 4], structural similarity [5], melodic similarity [6–8], harmonic similarity [9], lyrics similarity [10], timbral similarity, and similarity of performance characteristics [9, 11]. This list is far from exhaustive. Some of the listed characteristics will impact genre similarity, but others might not. McAdams et al. propose to group all types of similarity into three clusters: surface and texture, figural, and structural [12].

Note that the subjective nature of music similarity applies also to these aspects by themselves. That means that while rhythmic similarity obviously represents only one aspect of general music similarity, defining a model of rhythmic similarity is a difficult task in itself. Furthermore, there are indications that subjective music similarity ratings depend, among other things, on the familiarity of the test subject with the music [13] and possibly on the listener's expertise as well.

Another factor complicating measures of similarity is the sequential nature of music, i.e. the similarity between two pieces could vary depending on which time segments are compared. Last but not least, the associative nature of memory cannot be neglected in real-world applications. Editorial data and context will also influence a human's rating of similarity. Editorial data refers to data that cannot be extracted from the audio signal such as recording date and studio, artists and producers who participated in other albums, the label.

12.1.1 Approaches to Music Similarity Computation

The current knowledge about music similarity is insufficient to build reliable quantitative models; however, even the existing knowledge of music psychologists and empirical musicologists is rarely utilized for the design of systems modeling audio similarity. For example, despite the wide range of similarity dimensions mentioned above, the majority of work on measuring similarity of musical audio has been focusing on *timbre* similarity. The majority features used are therefore often basic timbre descriptors. They include, for example, Mel Frequency Cepstral Coefficients (MFCCs) [14–17], loudness-related, and rhythm-related features [18, 19]. Nowadays, basically all low-level features and mid-level features introduced in the preceding chapters have been utilized for the computation of music similarity [20, 21]. Other, less common features model the spectral envelope, for example through K-means clustering [15] or Gaussian Mixture Models (GMMs) [17]. Similarly, Pampalk et al. proposed a histogram containing level classes per frequency band [19].

The mapping of a music recording into the resulting, potentially high-dimensional, feature space is then used to model similarity by computing distance measures between this recording and others. Common pairwise distance measures are the Euclidean distance and the cosine distance (see Section 4.4). The choice of features and their relative weighting to each other, therefore, can be seen as an ad hoc definition of music similarity used by this system, while the choice of the distance measure aims at quantifying the perceptual impression of similarity.

12.1.2 Evaluation

Researchers noted very early the challenges in evaluating any attempts at music similarity estimation. Even when focusing on timbre similarity specifically, the unclear definition of the task impedes the generation of reliable ground truth for evaluation. Aucouturier and Pachet suggest three ways of evaluating a system [17]: (i) qualitative evaluation, i.e. looking and analyzing results for various pairs of songs, a method that Pampalk et al. built on by visualizing similarity distance through a Self-Organizing Map (SOM) [19], (ii) evaluation through annotations for a related task such as the genre, the artist, the album, or other tags [15, 19], and (iii) listening survey results that indicate the degree of similarity [22]. Flexer and Grill, however, point out that there is huge variance between test subjects ranking similarity and thus the ground truth may only be of limited use [23].

While the result of listening studies is ultimately the only way of reliably evaluating music similarity systems, the fundamental issues in the definition of this task and the accompanying challenges defining a proper methodology and setup of listening studies have ultimately led to slow progress in establishing ground truth data and evaluation methodology. One of the few publicly available datasets has only been released recently [24].

12.2 Musical Genre Classification

The automatic classification of audio signals and in particular the recognition of the musical genre or the *musical style* from an audio signal is one of the oldest subjects of Audio Content Analysis (ACA) and can be regarded as one of the seminal research topics shaping the research field.

The goal of *musical genre classification* is assigning a category from a predefined set of genre categories to a music recording based on the stylistic properties of the recorded piece. The task is a classic machine learning task – suitable features are extracted from the audio signal and with these features a classifier is trained and used for classification.

Applications for this technology can be found mainly in the annotation, sorting, recommendation, and retrieval of (similar) audio files from large databases or the Internet.

At first glance, the meaning of the term *musical genre* appears to be self-explanatory and its intuitive meaning obvious. On a more thorough investigation, however, it has to be concluded that an objective definition of the term is hardly possible. First, it should be noted that this text uses “genre” and “style” interchangeably, although it has been argued these terms have separate meanings [25]. Musical genres are labels assigned to categories grouping songs together based on some measure of similarity, tradition, and set of conventions. The similarity dimensions used to cluster music into these categories can include a large number of high-level and low-level musical characteristics, such as musical form (e.g. Blues), instrumentation (e.g. String Quartet), timbre and effects (e.g. distortion in Rock), rhythm (e.g. Reggae), and performance techniques (e.g. Rap), but it might also include sociological, text-based, or other measures of similarity such as epoch as well as lyrics content and style [25, 26]. Pachet and Cazaly [27] and later Scaringella et al. [28] summarize some issues with existing genre categorizations:

- *Scope of the genre label*: Can an individual song be classified into a genre or does the context of album and performing artist influence or overrule the classification decision? Or may even different parts of a piece of music have different genres?
- *Nonagreement of taxonomies*: The number and definition of genre labels can strongly vary; in the year 2000, Pachet and Cazaly compared genre taxonomies from the three web sites Allmusic, Amazon, and MP3.com. The overall number of genres varied from 430 (MP3.com) to 531 (Allmusic) to 719 (Amazon), and these labels had only 70 terms in common. Furthermore, the hierarchy of the taxonomies differed. In some practical applications of ACA, the number of genres has to be restricted due to technical limitations of the classification systems used, but the underlying problems of defining a taxonomy remain the same. Figure 12.1 shows two taxonomies defined in the context of early musical genre classification systems.
- *Ill-defined genre labels*: There is a semantic confusion between genre labels. They can be geographically defined (*Indian music*), related to an epoch in music history (*Baroque*), refer to technical requirements (*barbershop*), the instrumentation (*symphonic music*), or usage (*Christmas songs*).
- *Scalability of genre taxonomies*: A specific genre may be split into a variety of subgenres (e.g. *Hip Hop* into *Gangsta Rap*). The number of subgenres might evolve over the years.
- *Nonorthogonality of genre categories*: One piece of music can possibly be classified into multiple genre categories at the same time.
- *Incompleteness of genre categories*: A piece might fall between or outside of predefined genre categories.

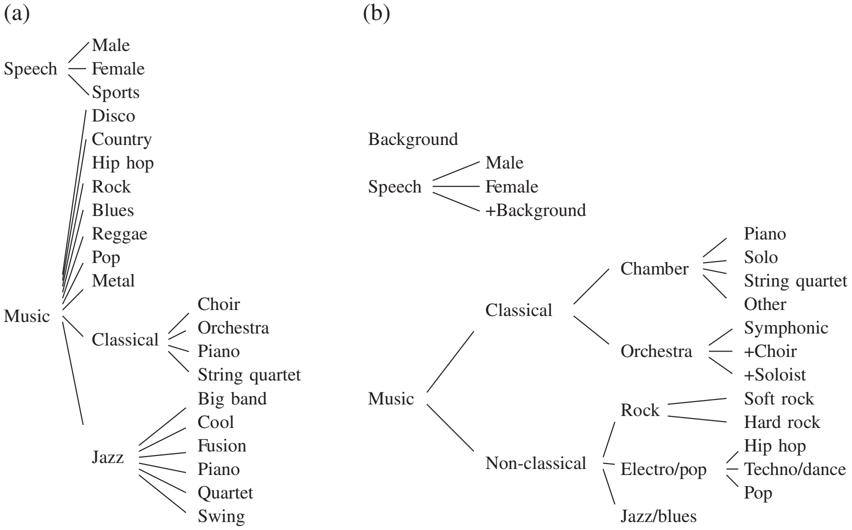

(a)
Speech
Male
Female
Sports
Music
Disco
Country
Hip hop
Rock
Blues
Reggae
Pop
Metal
Classical
Choir
Orchestra
Piano
String quartet
Jazz
Big band
Cool
Fusion
Piano
Quartet
Swing
(b)
Background
Speech
Male
Female
+Background
Music
Classical
Chamber
Piano
Solo
String quartet
Other
Orchestra
Symphonic
+Choir
+Soloist
Non-classical
Rock
Soft rock
Hard rock
Electro/pop
Hip hop
Techno/dance
Pop
Jazz/blues

Figure 12.1 Two examples for author-defined genre taxonomies. (a) Tzanetakis and Cook. Source: Modified from Tzanetakis and Cook [29], (b) Burred and Lerch. Source: Data from Burred and Lerch [30].

Given these observations, it becomes clear that deriving a complete and musicologically consistent taxonomy is practically impossible [30]. Researchers have also been making the case for music genre classification to be approached as a multilabel task allowing for multiple labels per input [31, 32]. Without a clear definition of the term *musical genre*, it also comes as no surprise that the performance of humans annotating pieces with genre labels is far from perfect [33–35]. Since ACA systems are trained with human-labeled data, automatic systems can be hardly expected to outperform humans at this task.

12.2.1 Approaches to Musical Genre Classification

The aforementioned issues in defining genre unambiguously and objectively pose fundamental challenges to the task of musical genre classification. For these reasons, researchers have even suggested abandoning all attempts at music classification and focusing on other tasks [36]. Nevertheless, genre classification remains a popular research topic in the field of MIR because of its appealing and intuitive task definition and the general availability of labeled data.

The first publications on the automatic recognition of musical style appeared in the 1990s; at that time, the focus was mainly on the discrimination of speech and music signals [37–41], although studies aiming at classifying a broader range of signals can be found during that decade as well [14, 42]. Tzanetakis and Cook introduced what has been for many years considered a standard baseline approach

[29]: low-level feature extraction and aggregation followed by multiclass classification. General surveys of approaches to musical genre classification have been published by Scaringella et al. [28] and Fu et al. [43].

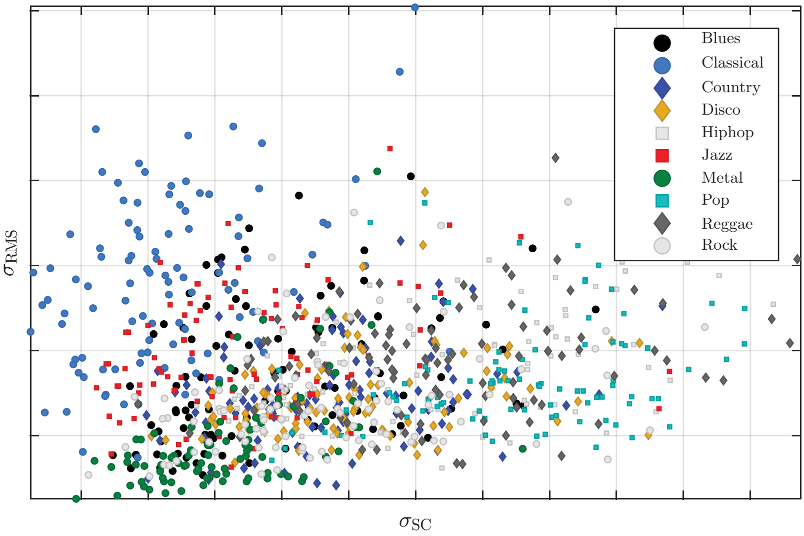
From a perceptual point of view, musical genre categorization may be influenced by

- *temporal characteristics* such as the tempo, the time signature, and rhythmic patterns,
- *dynamic characteristics* such as the loudness range, the change of loudness over time, accents,
- *tonal characteristics* such as melodic properties, the complexity of harmony (progression), and prominent pitch classes,
- *production-related characteristics* such as a specific sound quality and volume relations between instruments, effects and stereo panning properties, and
- *instrumentational characteristics* such as the number and type of instruments used.

Ultimately, this covers all musical audio content categories as introduced in Section 2.1 (temporal, dynamics, pitch-based, timbre) and indicates that a wide variety of features should be helpful for the task.

Early audio classification systems utilized only a restricted set of features, starting with simple features such as the zero crossing rate (see Section 3.5.13), counting the number of sign changes in the audio signal, and intensity-related features (see Chapter 8) such as the root mean square (RMS) [37, 38, 44, 45], although timbre-related spectral features such as the spectral centroid and the MFCCs quickly got added to the default set of features [14, 42, 46]. Figure 12.2 plots a simple feature space representation of the GTZAN dataset [29] spanned by the standard deviation of the spectral centroid (x-axis) and standard deviation of the RMS (y-axis). It can be observed that, while some loose clusters corresponding to the 10 music classes can be identified, these two features alone apparently do not allow for good class separation.

Over the years, more and more features were investigated for this task, eventually covering not only all features presented in Section 3.5 but also additional custom-designed instantaneous features [47–49]. Pitch-related features including properties of the fundamental frequency variation could also be found in the early speech/music classification systems [40, 50, 51] but are usually not used for systems targeting polyphonic audio input. Over the years, intensity and timbre features have consistently outperformed features from other domains, with the Mel Frequency Cepstral Coefficients (MFCCs) nowadays often considered the most powerful single (albeit multidimensional) feature. However, many features from other domains have been proposed. Researchers have investigated features derived from a beat histogram describing temporal and rhythmic content



Repo 1: plotFeatureScatter.m

Figure 12.2 Scatterplot of the feature space for the 10 music classes of the GTZAN dataset spanned by the standard deviation of the RMS and the spectral centroid, respectively.

(see Section 9.4) [29, 30, 52], simple tonal features such as features extracted from a pitch histogram [53], and production-related features such as stereo panning features [54].

Nowadays, these manually designed features are commonly replaced by learned representations as introduced in Section 3.6. This replacement can either happen through training a neural network with, e.g. a Mel Spectrogram input, or by utilizing transfer learning approaches. While such transferred features can be learned from music [55, 56], it is interesting to note that features learned for the classification of general audio such as VGGish [57] or OpenL3 [58] can perform well on music-related tasks [59].

As genre annotations usually apply to a complete song or recording, one could assume that the features mentioned above are simply aggregated over the whole audio file (see Section 3.7.2). However, the aggregation window (or texture window) can also be significantly shorter: Tzanetakis and Cook found that the classification accuracy does not improve after a texture window length of one second [29], although Ahrendt et al. found a window length of five seconds to be sufficient in most cases [60], and Burred and Lerch identified a texture window of approximately 15 s length to yield the best results [30]. No solid conclusions can be drawn from these results except that the optimal texture window size depends strongly either on the features and subfeatures used or on the data set and its categories. Texture windows have also been applied hierarchically by aggregating features over a short window and then in turn aggregating the aggregated results over the file [61]. It is interesting to note that humans seem to excel at quickly identifying the musical genre; in an experiment with a data set with 10 genres, subjects were able to identify the genre reliably after listening to audio snippets significantly shorter than 1 s [62].

Traditionally, typical classifiers are K-Nearest Neighbor (K-NN) [42], GMM [29, 30] and Support Vector Machine (SVM) [63]. A short description of these classifiers can be found in Section 4.1. A most common "default" traditional baseline approach is a SVM classifier with MFCC inputs.

While Artificial Neural Networks (ANNs) have been proposed for musical genre classification as early as the late 1990s [46], only the performance of more modern deep architectures proved superior to the classifiers mentioned above. A common deep learning architecture is a Convolutional Neural Network (CNN) with a Mel spectrogram input [64].

12.2.2 Genre Classification: Example

Pulling together content from various sections of this book, we have all the knowledge and tools to build a simple speech/music classifier from scratch to exemplify the basic processing chain of an audio classification task. For this example, we use a small speech/music dataset consisting of 64 audio recordings per class and attempt to classify with two basic instantaneous features. The processing steps are:

1. for each audio file
    - (a) read audio data from file,
    - (b) split audio data into overlapping blocks (see Section 3.2.5),
    - (c) compute the spectral centroid (see Section 3.5.1) and RMS (see Section 8.3.1) per block, and
    - (d) aggregate each feature by computing the mean of the spectral centroid and the standard deviation of the RMS, resulting in a two-dimensional vector for each audio file as shown in Figure 4.2 (see Section 3.7.2)
2. normalize both features across all files with z-score normalization (see Section 3.7.3)
3. apply a K-NN classifier with $k = 3$ in leave-one-out cross validation (see Sections 4 and 5.1)
4. compute the confusion matrix and the classification accuracy (see Section 6.1.1)

Listing 12.1 shows the implementation of these steps.

The scatter plot for the two extracted features is shown in Figure 4.2.

The resulting confusion matrix is shown in Table 12.1, leading to an overall accuracy of

$$\frac{51 + 53}{64 + 64} = 81.25\%.$$

```
% extract features, this may take a while...
for i = 1:size(music_files, 1)
    v_music(:, i) = ExtractFeaturesFromFile_I(...
        [cDatasetPath 'music/' music_files(i).name]);
end
for i = 1:size(speech_files, 1)
    v_speech(:, i) = ExtractFeaturesFromFile_I(...
        [cDatasetPath 'speech/' speech_files(i).name]);
end

% assign class labels for training and eval
C = [zeros(1, size(music_files, 1)) ones(1, size(speech_files, 1))];

% normalize features
v = [v_music, v_speech];
m = mean(v, 2);
s = std(v, 0, 2);
v = (v - repmat(m, 1, size(music_files, 1) + size(speech_files, 1)))./...
    repmat(s, 1, size(music_files, 1)+size(speech_files, 1));

% compute the overall accuracy with cross validation
[acc, mat] = ToolLooCrossVal(v, C);
```

Repo 2: ExampleMusicSpeechClassification.m

Listing 12.1 Example implementation for training and evaluating a simple speech/music classifier.

Table 12.1 Confusion matrix for an example evaluation of a speech/music classifier.

	Pred. speech	**Pred. music**	**# files**
GT speech	**51**	13	64
GT music	11	**53**	64

The correct (true) classifications are highlighted in bold.

The classification with only one of the two features would lead to the accuracies 63.28% and 73.44% for (the mean of the) spectral centroid and (the standard deviation of the) RMS, respectively.

12.2.3 Evaluation

As one of the historic core tasks in Music Information Retrieval (MIR), music genre classification is one of the most evaluated tasks [65]. As it is usually considered to be a straightforward multiclass classification task, the evaluation usually follows standard classification methodology. The biggest issues, as pointed out above, stem from the acquisition of data and their annotations and will be summarized below.

12.2.3.1 Metrics

The most common metrics are, unsurprisingly, the standard classification metrics introduced in Section 6.1.1: mean accuracy, precision, recall, and F-measure. The confusion matrix is also a common tool to analyze typical between-class confusions. As class imbalances are common in data, metrics should explicitly take that into account. It has also been suggested that the ground truth ambiguity inherent to this task should lead to adjustments in evaluation methodology and metrics [66].

12.2.3.2 Data

The long history of the task combined with the intuitive appeal of automating musical genre annotations and the availability of existing annotations resulted in a comparably large number of publicly available datasets. They are, however, not usually compatible as they vary considerably in their definition of genre taxonomy, e.g. in their labels, the number of genre classes, and their hierarchical structure. Many datasets also show strong preference for or focus exclusively on Western genres. Besides the genre label assignment itself, datasets have been criticized for including multiple songs of the same album or artist and thus possibly confounding the task with artist identification [67–69]. Typical problems with genre datasets have been highlighted in an in-depth analysis of the frequently-used GTZAN dataset (see below) by Sturm [70] and have also led to approaches measuring data validity [71].

The most commonly used datasets with available audio data are (listed alphabetically) the Ballroom dataset with 8 classes [72], the FMA datasets with up to 161 classes [73], the GTZAN dataset with 10 classes [29], the Homburg dataset with 9 classes [74], the ISMIR2004 genre dataset with 6 classes, the RWC dataset with 10 classes [75], and the Seyerlehner datasets with up to 19 classes [76].

12.2.3.3 Results

Parsing the MIREX results over the past few years,[1] accuracies roughly in the range of 50–70% can be observed for tasks with 7–10 classes. While these numbers are mostly meaningless without detailed information on the audio collection and annotation process, at least they can provide a ballpark number for what to expect when working with genre data.

12.2.4 Exercises

12.2.5 Questions

1. How does the subjectivity of the definition of *genre* impact the task of genre classification?
2. List three inconsistencies in common genre taxonomies.

---

[1] MIREX. https://www.music&hyphen;ir.org/mirex. Last retrieved on 8 February 2022.

12.2.6 Assignments

This assignment brings together many functions that we have implemented in previous assignments.

1. Download the GTZAN genre collection dataset.[2]
2. Feature Extraction: use the functions
   - `extract_features`,
   - `aggregate_feature_per_file`, and
   - `normalize_zscore`

   from Section 3.8.2 to extract a normalized feature matrix from the dataset and create the vector with ground truth labels.
3. Cross-Validation: use the function
   - `cross_validate`

   from Section 5.6 and rank the single best feature using a threefold cross validation using the above method from a newly implemented function called `[feature_index] = find_best_features(data, labels, k, num_folds)`. This function calls your `cross_validate` with `num_folds` = 3, `k` = 3 for all the features individually and finds the feature which performs the best. Report your result.
4. Feature Selection
   - Write a function `[sel_feature_ind, accuracy] = select_features(data, gt_labels, k, num_folds)` which computes the best feature set based on a sequential forward selection process. `data` is the feature matrix as np arrays with the dimensions numOfFeatures × numOfObservations, `gt_labels` are the ground truth class indices (numOfObservations), accuracy is a vector containing the accuracy given the number of selected features (accuracy with the best feature, the best feature pair, etc.). Run your feature selection `num_folds` = 3, `k` = 3. Report the best order of features. Plot how the accuracy changes with the feature selection iterations.
5. Evaluation
   - Using the best set of features (highest accuracy) obtained above, run your `cross_validate` for different values of `k` = 1, 3 and 7. Use `num_folds` = 10 for this. Implement this in a function `[accuracies, conf_matrices] = evaluate(data, labels)`. Report the three results of your cross validation (accuracies and confusion matrices). Discuss the confusion matrices obtained in terms of what genres are confused with each other. How does k affect the classification performance?

---

[2] GTZAN. href="http://marsyas.info/downloads/datasets.html. Last retrieved on 27 November 2021.

6. Comparison to unsupervised clustering: use the function
   - `kmeans`

   from Section 5.6 and perform your K-Means clustering using `k = 10` for the genre data provided to you. Contrast the cluster labels with the ground truth genre labels. Discuss your results in terms of how closely genre relates to music similarity (based on this particular set of the features).

References

1 Beth Logan. Content-based Playlist Generation: Exploratory Experiments. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Paris, 2002.

2 Malcolm Slaney. Web-Scale Multimedia Analysis: Does Content Matter? *IEEE Multimedia*, 18(2):12–15, February 2011. ISSN 1070-986X. doi: 10.1109/MMUL.2011.34. URL http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5754643.

3 Mark A Pitt and Caroline B Monahan. The Perceived Similarity of Auditory Polyrhythms. *Perception & Psychophysics*, 41(6):534–546, June 1987. ISSN 0031-5117. URL http://www.ncbi.nlm.nih.gov/pubmed/3615150.

4 Ludger Hofmann-Engl. Rhythmic Similarity: A Theoretical and Empirical Approach. In *Proceedings of the 7th International Conference on Music Perception and Cognition (ICMPC)*, pages 564–567, Sydney, 2002.

5 Emmanuel Bigand. Abstraction of Two Forms of Underlying Structure in a Tonal Melody. *Psychology of Music*, 18(1):45–59, April 1990. ISSN 0305-7356. doi: 10.1177/0305735690181004. URL http://pom.sagepub.com/cgi/doi/10.1177/0305735690181004.

6 James C Bartlett. Scale Structure and Similarity of Melodies. *Music Perception*, 5(3):285–314, 1988.

7 Tuomas Eerola, Topi Järvinen, Jukka Louhivuori, and Petri Toiviainen. Statistical Features and Perceived Similarity of Folk Melodies. *Music Perception*, 18(3):275–296, March 2001. ISSN 0730-7829. doi: 10.1525/mp.2001.18.3.275. URL http://www.jstor.org/stable/40285837.

8 Greg Aloupis, Thomas Fevens, Stefan Langerman, Tomomi Matsui, Antonio Mesa, Yurai Nunez, David Rappaport, and Godfried T Toussaint. Algorithms for Computing Geometric Measures of Melodic Similarity. *Computer Music Journal*, 30(3):67–76, 2006. ISSN 1531-5169. URL https://muse.jhu.edu/article/202595.

9 Alexandra Lamont and Nicola Dibben. Motivic Structure and the Perception of Similarity. *Music Perception*, 18(3):245–274, March 2001. ISSN 0730-7829. doi: 10.1525/mp.2001.18.3.245. URL http://www.jstor.org/stable/40285836.

10 Beth Logan, Andrew Kositsky, and Pedro Moreno. Semantic Analysis of Song Lyrics. In *Proceedings of the International Conference on Multimedia and Expo (ICME)*, volume 2, pages 827–830 Vol.2, Taipei, Taiwan, June 2004. doi: 10.1109/ICME.2004.1394328.

11 Renee Timmers. Predicting the Similarity Between Expressive Performances of Music from Measurements of Tempo and Dynamics. *Journal of the Acoustical Society of America (JASA)*, 117(1):391–399, January 2005. ISSN 0001-4966. doi: 10.1121/1.1835504. URL https://asa.scitation.org/doi/10.1121/1.1835504.

12 Stephen McAdams and Daniel Matzkin. Similarity, Invariance, and Musical Variation. *Annals of the New York Academy of Sciences*, 930(1):62–76, January 2001. ISSN 00778923. doi: 10.1111/j.1749-6632.2001.tb05725.x. URL http://www.ncbi.nlm.nih.gov/pubmed/11458867 http://doi.wiley.com/10.1111/j.1749-6632.2001.tb05725.x.

13 Alberto Novello, Martin F Mckinney, and Armin Kohlrausch. Perceptual Evaluation of Music Similarity. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Victoria, 2007.

14 Jonathan T Foote. A Similarity Measure for Automatic Audio Classification. In *Proceedings of the 14th National Conference on Artificial Intelligence*, Providence, RI, 1997.

15 Beth Logan and Ariel Salomon. A Music Similarity Function based on Signal Analysis. In *Proceedings of the International Conference on Multimedia and Expo (ICME)*, pages 745–748, 2001. doi: 10.1109/ICME.2001.1237829. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1237829.

16 Jonathan T Foote and Matthew Cooper. Visualizing Musical Structure and Rhythm via Self-Similarity. In *Proceedings of the International Computer Music Conference (ICMC)*, Habana, September 2001.

17 Jean-Julien Aucouturier and Francois Pachet. Music Similarity Measures: What's the Use? In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Paris, 2002.

18 Elias Pampalk, Andreas Rauber, and Dieter Merkl. Content-based Organization and Visualization of Music Archives. *Proceedings of the 10th ACM International Conference on Multimedia*, page 570, 2002. doi: 10.1145/641007.641121. URL http://portal.acm.org/citation.cfm?doid=641007.641121.

19 Elias Pampalk, Simon Dixon, and Gerhard Widmer. On the Evaluation of Perceptual Similarity Measures for Music. In *Proceedings of the 6th International Conference on Digital Audio Effects (DAFX)*, London, September 2003.

20 Tao Li and Mitsunori Ogihara. Content-based Music Similarity Search and Emotion Detection. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 5, pages 705–708. IEEE, 2004. ISBN 0-7803-8484-9. doi: 10.1109/ICASSP.2004.1327208. URL http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1327208.

21 Dmitry Bogdanov, Joan Serrà, Nicolas Wack, and Perfecto Herrera. From Low-Level to High-Level: Comparative Study of Music Similarity Measures. *Proceedings of the 11th International Symposium on Multimedia*, pages 453–458, 2009. doi: 10.1109/ISM.2009.72. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5366050.

22 Beth Logan, Daniel P W Ellis, and Adam Berenzweig. Toward Evaluation Techniques for Music Similarity. In *Proceedings of the Workshop on the Evaluation of Music Information Retrieval System at SIGIR*, Toronto, 2003.

23 Arthur Flexer and Thomas Grill. The Problem of Limited Inter-rater Agreement in Modelling Music Similarity. *Journal of New Music Research*, 45(3):239–251, July 2016. ISSN 0929-8215, 1744-5027. doi: 10.1080/09298215.2016.1200631. URL https://www.tandfonline.com/doi/full/10.1080/09298215.2016.1200631.

24 Jongpil Lee, Nicholas J Bryan, Justin Salamon, Zeyu Jin, and Juhan Nam. Disentangled Multidimensional Metric Learning for Music Similarity. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6–10, May 2020. doi: 10.1109/ICASSP40776.2020.9053442. ISSN: 2379-190X.

25 Allan F Moore. Categorical Conventions in Music Discourse: Style and Genre. *Music & Letters*, 82(3):432–442, 2001. ISSN 0027-4224. URL https://www.jstor.org/stable/3526163. Publisher: Oxford University Press.

26 Roger B Dannenberg. Style in Music. In Shlomo Argamon, Kevin Burns, and Shlomo Dubnov, editors, *The Structure of Style: Algorithmic Approaches to Understanding Manner and Meaning*, pages 45–57. Springer-Verlag, Berlin, Heidelberg, 2010. ISBN 978-3-642-12337-5. URL https://doi.org/10.1007/978-3-642-12337-5.

27 Francois Pachet and Daniel Cazaly. A Taxonomy of Musical Genres. In *Proceedings of the Conference on Content-Based Multimedia Information Access*, Paris, April 2000. 00219.

28 Nicolas Scaringella, Giorgio Zoia, and Daniel Mlynek. Automatic Genre Classification of Music Content: A Survey. *Signal Processing Magazine*, 23(2):133–141, 2006.

29 George Tzanetakis and Perry Cook. Musical Genre Classification of Audio Signals. *Transactions on Speech and Audio Processing*, 10(5):293–302, July 2002. ISSN 1063-6676. doi: 10.1109/TSA.2002.800560. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1021072.

30 Juan José Burred and Alexander Lerch. Hierarchical Automatic Audio Signal Classification. *Journal of the Audio Engineering Society (JAES)*, 52(7/8):724–739, 2004. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2016/10/Burred-and-Lerch-2004-Hierarchical-Automatic-Audio-Signal-Classification.pdf.

31 Hanna Lukashevich, Jakob Abeßer, Christian Dittmar, and Holger Grossmann. From Multi-Labeling to Multi-Domain-Labeling: A Novel Two-Dimensional Approach to Music Genre Classification. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 6, Kobe, Japan, 2009.

32 Chris Sanden and John Z Zhang. Enhancing Multi-Label Music Genre Classification through Ensemble Techniques. In *Proceedings of the International Conference on Research and Development in Information Retrieval (SIGIR)*, SIGIR '11, pages 705–714, New York, NY, USA, July 2011. Association for Computing Machinery. ISBN 978-1-4503-0757-4. doi: 10.1145/2009916.2010011. URL https://doi.org/10.1145/2009916.2010011.

33 Hagen Soltau. *Erkennung von Musikstilen*. Diploma Thesis, Universität Karlsruhe, Karlsruhe, 1997.

34 Stefaan Lippens, Jean-Pierre Martens, Tom De Mulder, and George Tzanetakis. A Comparison of Human and Automatic Musical Genre Classification. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Montreal, 2004.

35 Klaus Seyerlehner, Gerhard Widmer, and Peter Knees. A Comparison of Human, Automatic and Collaborative Music Genre Classification and User Centric Evaluation of Genre Classification Systems. In *Proceedings of the International Workshop on Adaptive Multimedia Retrieval. Context, Exploration, and Fusion*, Lecture Notes in Computer Science, pages 118–131, Berlin, Heidelberg, 2010. Springer. ISBN 978-3-642-27169-4. doi: 10.1007/978-3-642-27169-4_9.

36 Cory McKay and Ichiro Fujinaga. Musical Genre Classification: Is it Worth Pursuing and How can it be Improved? In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 101–106, Victoria, 2006.

37 John Saunders. Real-Time Discrimination of Broadcast Speech/Music. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Atlanta, 1996.

38 Eric D Scheirer and Malcolm Slaney. Construction and Evaluation of a Robust Multifeature Speech/Music Discriminator. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Munich, April 1997.

39 Gethin Williams and Daniel P W Ellis. Speech/Music Discrimination Based on Posterior Probability Features. In *Proceedings of the 6th European Conference on Speech Communication and Technology (EUROSPEECH)*, Budapest, September 1999. 00105.

40 Michael J Carey, Eluned S Parris, and Harvey Lloyd-Thomas. A Comparison of Features for Speech, Music Discrimination. In *Proceedings of the International*

Conference on Acoustics, Speech, and Signal Processing (ICASSP), Phoenix, 1999.

41 Khaled El-Maleh, Mark Klein, Grace Petrucci, and Peter Kabal. Speech/Music Discrimination for Multimedia Applications. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Istanbul, 2000.

42 Erling Wold, Thom Blum, Douglas Keislar, and James Wheaten. Content-based Classification, Search, and Retrieval of Audio. *IEEE Multimedia*, 3(3):27–36, 1996. ISSN 1070986X. doi: 10.1109/93.556537. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=556537.

43 Zhouyu Fu, Guojun Lu, Kai Ming Ting, and Dengsheng Zhang. A Survey of Audio-Based Music Classification and Annotation. *Transactions on Multimedia*, 13(2):303–319, April 2011. ISSN 1520-9210. doi: 10.1109/TMM.2010.2098858. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5664796.

44 Tryphon Lambrou, Panos Kudumakis, Robert D Speller, Mark B Sandler, and Alf D Linney. Classification of Audio Signals using Statistical Features on Time and Wavelet Transform Domains. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 6, pages 3621–3624, 1998. doi: 10.1109/ICASSP.1998.679665. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=679665.

45 Zhu Liu, Yao Wang, and Tsuhan Chen. Audio FFeature Extraction and Analysis for Scene Classification. *Journal of VLSI Signal Processing*, pages 343–348, 1998. doi: 10.1109/MMSP.1997.602659.

46 Hagen Soltau, Tanja Schulz, Martin Westphal, and Alex Waibel. Recognition of Music Types. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Seattle, 1998.

47 Stan Z Li. Content-based Audio Classification and Retrieval using the Nearest Feature Line Method. *Transactions on Speech and Audio Processing*, 8(5):619–625, 2000. ISSN 10636676. doi: 10.1109/89.861383. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=861383.

48 Martin F McKinney and Jeroen Breebart. Features for Audio and Music Classification. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Baltimore, MD, 2003.

49 Juan José Burred and Alexander Lerch. A Hierarchical Approach to Automatic Musical Genre Classification. In *Proceedings of the 6th International Conference on Digital Audio Effects (DAFX)*, London, September 2003. doi: 10.1.1.2.6582. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2016/10/Burred-and-Lerch-2003-A-Hierarchical-Approach-to-Automatic-Musical-Genre.pdf.

50 Roger B Dannenberg, Belinda Thom, and David Watson. A Machine Learning Approach to Musical Style Recognition. In *Proceedings of the International Computer Music Conference (ICMC)*. Thessaloniki, September 1997.

51 Tong Zhang. Hierarchical System for Content-based Audio Classification and Retrieval. *Proceedings of SPIE*, pages 398–409, 1998. ISSN 0277786X. doi: 10.1117/12.325832. URL http://link.aip.org/link/?PSI/3527/398/1&Agg=doi.

52 Simon Dixon, Elias Pampalk, and Gerhard Widmer. Classification of Dance Music by Periodicity Patterns. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 1–7, 2003.

53 George Tzanetakis, Andrey Ermolinskyi, and Perry Cook. Pitch Histograms in Audio and Symbolic Music Information Retrieval. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Paris, 2002.

54 George Tzanetakis, Randy Jones, and Kirk McNally. Stereo Panning Features for Classifying Recording Production Style. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 441–444, Vienna, 2007. URL http://webhome.cs.uvic.ca/ gtzan/work/pubs/ismir07gtzanE.pdf.

55 Keunwoo Choi, György Fazekas, Mark B Sandler, and Kyunghyun Cho. Transfer Learning for Music Classification and Regression Tasks. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 141–149, Suzhou, China, 2017. URL https://ismir2017.smcnus.org/wp-content/uploads/2017/10/12_Paper.pdf.

56 Jordi Pons and Xavier Serra. musicnn: Pre-Trained Convolutional Neural Networks for Music Audio Tagging. In *Late Breaking Demo (Extended Abstract), Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 2, Delft, Netherlands, 2019. URL http://arxiv.org/abs/1909.06654.

57 Shawn Hershey, Sourish Chaudhuri, Dan P W Ellis, Jort F Gemmeke, Aren Jansen, R Channing Moore, Manoj Plakal, Devin Platt, Riff A Saurous, Bryan Seybold, Malcolm Slaney, Ron J Weiss, and Kevin Wilson. CNN Architectures for Large-Scale Audio Classification. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 131–135, New Orleans, LA, March 2017. Institute of Electrical and Electronics Engineers (IEEE). doi: 10.1109/ICASSP.2017.7952132. ISSN: 2379-190X.

58 Jason Cramer, Ho-Hsiang Wu, Justin Salamon, and Juan Pablo Bello. Look, Listen, and Learn More: Design Choices for Deep Audio Embeddings. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3852–3856, Brighton, UK, May 2019. Institute of

Electrical and Electronics Engineers (IEEE). ISBN 978-1-4799-8131-1. doi: 10.1109/ICASSP.2019.8682475. URL https://ieeexplore.ieee.org/document/8682475/.

59 Sascha Grollmisch, Estefanía Cano, Christian Kehling, and Michael Taenzer. Analyzing the Potential of Pre-Trained Embeddings for Audio Classification Tasks. In *Proceedings of the European Signal Processing Conference (EUSIPCO)*, pages 790–794, Online, 2020. doi: 10.23919/Eusipco47968.2020.9287743. ISSN: 2076-1465.

60 Peter Ahrendt, Anders Meng, and Jan Larsen. Decision Time Horizon for Music Genre Classification Using Short Time Features. In *Proceedings of the 12th European Signal Processing Conference (EUSIPCO)*, Vienna, September 2004.

61 George Tzanetakis and Perry Cook. MARSYAS: A Framework for Audio Analysis. *Organised Sound*, 4(3):169–175, 2000.

62 Robert O Gjerdingen and David Perrott. Scanning the Dial: The Rapid Recognition of Music Genres. *Journal of New Music Research*, 37(2):93–100, June 2008. ISSN 0929-8215. doi: 10.1080/09298210802479268. URL http://www.tandfonline.com/doi/abs/10.1080/09298210802479268.

63 Guodong Guo and Stan Z Li. Content-based Audio Classification and Retrieval by Support Vector Machines. *Transactions on Neural Networks*, 14(1):209–215, January 2003. ISSN 1045-9227. doi: 10.1109/TNN.2002.806626. URL http://www.ncbi.nlm.nih.gov/pubmed/18238003.

64 Siddharth Sigtia and Simon Dixon. Improved Music Feature Learning with Deep Neural Networks. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6959–6963, Florence, Italy, May 2014. Institute of Electrical and Electronics Engineers (IEEE). ISBN 978-1-4799-2893-4. doi: 10.1109/ICASSP.2014.6854949. URL http://ieeexplore.ieee.org/document/6854949/.

65 Bob L Sturm. A Survey of Evaluation in Music Genre Recognition. In Andreas Nürnberger, Sebastian Stober, Birger Larsen, and Marcin Detyniecki, editors, *Adaptive Multimedia Retrieval: Semantics, Context, and Adaptation*, Lecture Notes in Computer Science, pages 29–66, Cham, 2014. Springer International Publishing. ISBN 978-3-319-12093-5. doi: 10.1007/978-3-319-12093-5_2.

66 Bob L Sturm. Classification Accuracy is Not Enough: On the Evaluation of Music Genre Recognition Systems. *Journal of Intelligent Information Systems*, 41(3):371–406, December 2013. ISSN 1573-7675. doi: 10.1007/s10844-013-0250-y. URL https://doi.org/10.1007/s10844-013-0250-y.

67 Elias Pampalk, Arthur Flexer, and Gerhard Widmer. Improvements of Audio-Based Music Similarity and Genre Classification. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 8, London, UK, 2005.

68 Arthur Flexer. A Closer Look on Artist Filters for Musical Genre Classification. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Vienna, 2007.

69 Arthur Flexer and Dominik Schnitzer. Effects of Album and Artist Filters in Audio Similarity Computed for Very Large Music Databases. *Computer Music Journal*, 34(3):20–28, 2010. URL http://www.mitpressjournals.org/doi/pdf/10.1162/COMJ_a_00004.

70 Bob L Sturm. An Analysis of the GTZAN Music Genre Dataset. In *Proceedings of the 2nd International ACM Workshop on Music Information Retrieval with User-Centered and Multimodal Strategies (MIRUM)*, Nara, 2012.

71 Yen-Cheng Lu, Chih-Wei Wu, Chang-Tien Lu, and Alexander Lerch. An Unsupervised Approach to Anomaly Detection in Music Datasets. In *Proceedings of the ACM SIGIR Conference (SIGIR)*, SIGIR '16, pages 749–752, Pisa, 2016. ACM. ISBN 978-1-4503-4069-4. doi: 10.1145/2911451.2914700. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2016/07/Lu-et-al_2016_An-Unsupervised-Approach-to-Anomaly-Detection-in-Music-Datasets.pdf.

72 Fabien Gouyon, Anssi Klapuri, Simon Dixon, Miguel Alonso, George Tzanetakis, Christian Uhle, and Pedro Cano. An Experimental Comparison of Audio Tempo Induction Algorithms. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(5):1832–1844, September 2006. ISSN 1558-7924. doi: 10.1109/TSA.2005.858509.

73 Michaël Defferrard, Kirell Benzi, Pierre Vandergheynst, and Xavier Bresson. FMA: A Dataset For Music Analysis. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Suzhou, China, September 2017. URL http://arxiv.org/abs/1612.01840.

74 Aline Honingh and Rens Bod. Clustering and Classification of Music by Interval Categories. In *Proceedings of the International Conference on Mathematics and Computation in Music*, *Lecture Notes in Computer Science*, pages 346–349, Berlin, Heidelberg, 2011. Springer. ISBN 978-3-642-21590-2. doi: 10.1007/978-3-642-21590-2_30.

75 Masataka Goto. RWC Music Database: Popular, Classical, and Jazz Music Databases. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 2, Paris, 2002.

76 Klaus Seyerlehner, Gerhard Widmer, and Peter Knees. Frame Level Audio Similarity - A Codebook Approach. In *Proceedings of the International Conference on Digital Audio Effects (DAFX)*, page 8, Espoo, Finland, 2008.

13

Mood Recognition

The mood of a piece of music is one of the cues a user often finds helpful for describing, finding, and browsing music [1]. This has led to a considerable amount of research targeting the automatic recognition of emotional characteristics of recordings of music. Common names for this task predicting the mood of a specific musical audio snippet are (audio) *mood recognition* and *music emotion recognition*. Albeit somewhat outdated, two overview articles provide a good introduction into the literature on this task [2, 3].

The synonymous use of the terms *emotion* and *mood* is a first indicator that this task definition faces similar or worse challenges than the definition of musical genre classification task introduced in Section 12.2. There is no general consensus across researchers working on this task on how emotion and mood are differentiated. It turns out that this is not only true for Audio Content Analysis (ACA) researchers: Kleinginna and Kleinginna, for example, reviewed more than a hundred scientific definitions of emotion [4] without being able to identify a consensus. They conclude that some definitions might be better or worse, but ultimately it is just not possible to prove the correctness of any individual definition. Weld described the difference between emotion and mood by characterizing emotion as temporary and evanescent in contrast to mood which is more permanent and stable [5]; mood has also been described as a diffuse affect state which can emerge without apparent cause [6]. However, in a musical context, it remains arguable if and when the term *emotion* should be preferred over the term *mood*.

Researching the relation of emotion and music exposes some of the typical problems of psychological research; one problem is, for example, the process of verbalization which confines the description of subtle and varied emotional states to the standardized words used to denote them [7]. Meyer points out that descriptions of emotions are usually apocryphal and misleading since emotions are named and distinguished largely in terms of the external circumstances in which the response takes place; music itself, however, presents no external circumstances [7]. Scherer criticizes the tendency to assume that music evokes basic emotions such as anger

An Introduction to Audio Content Analysis: Music Information Retrieval Tasks & Applications,
Second Edition. Alexander Lerch.
© 2023 The Institute of Electrical and Electronics Engineers, Inc. Published 2023 by John Wiley & Sons, Inc.

and fear [8]. This led him to propose the differentiation between *utilitarian* emotions, which are the emotions usually studied in emotion research (anger, fear, joy, disgust, sadness, shame, guilt, etc.), and *aesthetic* emotions which are not driven by external influences or personal goals but rather by the appreciation of the intrinsic qualities of a work of art [9]. Zentner et al. found that negative emotions such as guilt, shame, disgust, anger, fear are practically never aroused by music [10]. They also find that the "description of musical emotions requires a more nuanced affect vocabulary and taxonomy than is provided by current scales and models of emotion" [10].

The exact musical characteristics that lead to defining the mood of a piece of music continue being investigated. There is strong evidence of a relation between moods and both the tempo and the loudness of the performance, as reported by Juslin [11], Kantor-Martynuska [12], Sloboda and Lehmann [13], Schubert [14], and Timmers et al. [15]. The mode (*Major* vs. *Minor*) has also been reported to have influence on the perception of mood [16].

When discussing emotion and music, it is of importance to distinguish between the emotion aroused in the listener and the conveyed emotion perceived by a listener without necessarily feeling it [7]. Ratings of *perceived* emotion can differ significantly from ratings of *felt* emotion [10]. A performance projecting "happiness," for example, does not necessarily make the listener happy even though they may clearly perceive the projected emotion.

To complicate matters further, it might be of interest to differentiate between score-inherent emotions and performance-inherent emotions [17]. To study this, however, seems to be only possible in very controlled environments [11, 18, 19]; a real-world scenario does not allow for easy differentiation between emotions induced by score vs. performance properties, as the performance is an integral part of the "music" and thus cannot easily be investigated separately (see also Chapter 15). Furthermore, the differentiation between score and music performance is less distinct for popular music than for Western classical music.

13.1 Approaches to Mood Recognition

Building a model predicting mood from audio requires either a categorical or a quantitative representation of mood. A categorical representation assumes a finite set of distinct mood classes. Based on preceding research on measuring emotional response to music, Schubert grouped mood labels into nine clusters as shown in Table 13.1 [20]. Deriving mood clusters by statistical analysis from large publicly available metadata collections has been attempted by Hu and Downie [21]. The resulting five clusters as shown in Table 13.2 are also used in the Music Information Retrieval Evaluation eXchange (MIREX) automatic mood recognition task. These clusters can then be used for labeling the training data of a classifier.

Table 13.1 Mood clusters as presented by Schubert.

Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6	Cluster 7	Cluster 8	Cluster 9
Bright	Humorous	Calm	Dreamy	Dark	Heavy	Tragic	Agitated	Dramatic
Cheerful	Light	Delicate	Sentiment	Depressing	Majestic	Yearning	Angry	Exciting
Happy	Lyrical	Graceful		Gloomy	Sacred		Restless	Exhilarated
Joyous	Merry	Quiet		Melancholy	Serious		Tense	Passionate
	Playful	Relaxed		Mournful	Spiritual			Sensational
		Serene		Sad	Vigorous			Soaring
		Soothing		Solemn				Triumphant
		Tender						
		Tranquil						

Table 13.2 Mood clusters derived from metadata and used in MIREX.

Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
Rowdy	Amiable/good natured	Literate	Witty	Volatile
Rousing	Sweet	Wistful	Humorous	Fiery
Confident	Fun	Bittersweet	Whimsical	Visceral
Boisterous	Rollicking	Autumnal	Wry	Aggressive
Passionate	Cheerful	Brooding	Campy	Tense/Anxious
		Poignant	Quirky	Intense
			Silly	

Alternatively, musical mood is frequently represented as a model with two dimensions or parameters instead of distinct label categories. Russell's two-dimensional emotion space is one example of such models [22]. It describes mood in the two dimensions *pleasure-misery* (horizontal) and *arousal-sleepiness* (vertical). Figure 13.1 shows the resulting valence/arousal space. Using such a model, each mood will be represented by a two-dimensional (or even three-dimensional) coordinate in the mood space in the training data and can be predicted by a regression model. There have been other similar two-dimensional mood models presented [23, 24]; while they are not identical, they have some correspondence [25–27]. Researchers have been calling for adding a third dimension (for example, *dominance*, *tension*, or *interest*) to this model, but the usefulness of this third dimension is less obvious than for the first two dimensions [28, 29].

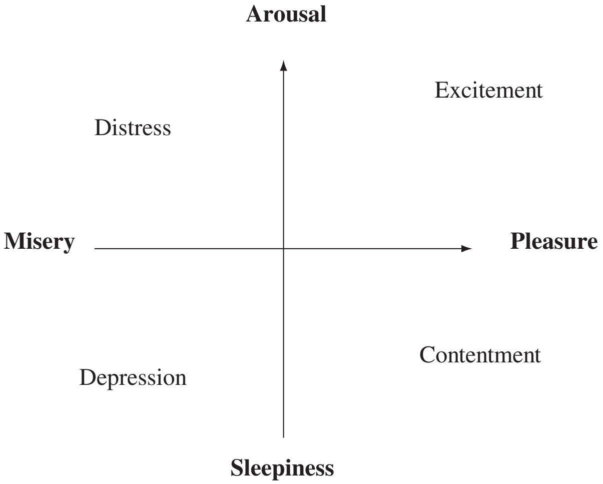
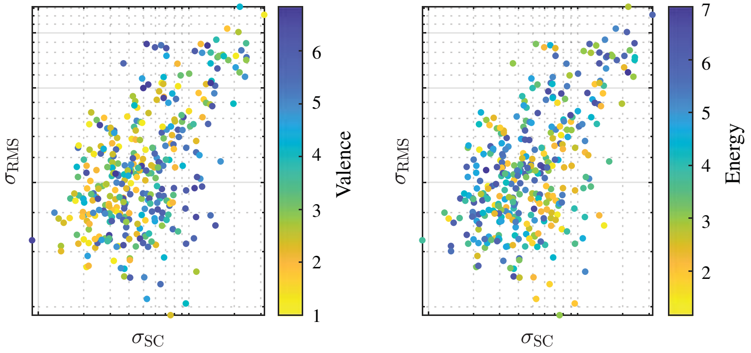


Figure 13.1 Russell's two-dimensional model of mood.

The feature extraction and classification algorithms used for mood recognition are very similar to the ones used in musical genre classification. The feature set often includes low-level features such as the Mel Frequency Cepstral Coefficients (MFCCs). An example of proposed mood-specific features which are not common in musical genre classification are articulation-based features estimating the smoothness of "note transitions" [30]. Mood recognition systems also use tempo, key, and rhythm-related features more frequently than systems for musical genre classification [31–34]. Since the relation between measurable characteristics in music and reported mood remains fuzzy and intangible, research is often exploratory in the sense that all conceivable features are investigated for the power to estimate musical mood given a specific set of ground truth annotations. A typical baseline classification and regression algorithm is the Support Vector Machine (SVM) and Support Vector Regression (SVR), respectively.

The complexity of this task is showcased in Figure 13.2, which displays the feature space spanned by the standard deviation of the spectral centroid (x-axis) and the standard deviation of the Root Mean Square (RMS) for the Soundtracks dataset [35]. Here, the *valence* and *energy* are rated on a scale of 1–7 and are color-coded on the left and right plot, respectively. If there were a monotonic relationship of one of these two features to one of the labels, a trajectory from dark to bright should be observable somewhere in these plots.

The definition and number of mood classes varies across studies. Feng et al., for example, classify music into the four classes *happiness*, *anger*, *sadness*, and *fear* [31], while Li and Ogihara uses the three dimensions *cheerful-depressing*, *relaxing-exciting*, and *comforting-disturbing* [36]. Russell's two-dimensional



Repo 1: plotFeatureScatterMood.m

Figure 13.2 Scatterplot of the feature space for the valence/energy ratings of the Soundtracks dataset spanned by the standard deviation of the RMS and the spectral centroid, respectively.

arousal/valence-model as shown in Figure 13.1 is probably the most frequently used model [22]. This model avoids using a categorical taxonomy by representing mood in continuous values for arousal and valence, respectively; each piece of music can be represented by a point in the two-dimensional space. The two values can then be predicted by regression algorithms [37, 38]. Some researchers, however, convert this model back to a categorical model by using each quadrant as a category or a cluster (*contentment*, *depression*, *exuberance*, *anxious/frantic*) [32, 39]. Other options are using a fuzzy classification approach that assigns probabilities to each class [39] and a multilabel classification system which assigns a group of labels to a test sample [32, 36, 40].

Note that most approaches mentioned above assume the mood label to be stationary for the whole input audio recording. It has been pointed out that this is not necessarily the case, triggering research trying to predict a temporal evolution of mood [32, 41]. The subjectivity of musical affect can also be taken into account by personalizing the model of emotional responses of individuals or groups [33].

13.2 Evaluation

While the issues in task subjectivity complicate the collection of trustworthy ground truth data, once these data have been accepted as ground truth, the system evaluation can be carried out mostly without further complications.

When mood recognition is interpreted as a classification task, the common metrics are accuracy and F-measure, precision, and recall. When it is interpreted as a regression task, the coefficient of determination R^2 is most commonly applied, but other metrics such as the mean average error (MAE) have been used [38].

For a introduction of these metrics please see Section 6.1.1. To evaluate both valence and arousal predictions combined, researchers have used the Euclidean distance [42].

The available datasets including audio data include the DEAM dataset [43], the Soundtracks dataset [35], and the EmoMusic dataset [44] with annotations for valence and arousal, and the TAFFC dataset [45] with quadrant annotation. Some of the available datasets come without audio files, e.g. the AMG1608 [42].

Recent results suggest that the accuracy range for classifying mood in the four quadrants or Russell's model is about 60–80% [46].

References

1 Fabio Vignoli. Digital Music Interaction Concepts: A User Study. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Barcelona, 2004.

2 Youngmoo E Kim, Erik M Schmidt, Raymond Migneco, Brandon G Morton, Patrick Richardson, Jeffrey Scott, Jacquelin A Speck, and Douglas Turnbull. Music Emotion Recognition: A State of the Art Review. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Utrecht, 2010.

3 Yi-Hsuan Yang and Homer H Chen. Machine Recognition of Music Emotion: A Review. *ACM Transactions on Intelligent Systems and Technology*, 3(3):40:1–40:30, May 2012. ISSN 2157-6904. doi: 10.1145/2168752.2168754. URL https://doi.org/10.1145/2168752.2168754.

4 Paul R Kleinginna and Anne M Kleinginna. A Categorized List of Emotion Definitions, with Suggestions for a Consensual Definition. *Motivation and Emotion*, 5(4):345–379, December 1981. ISSN 0146-7239. doi: 10.1007/BF00992553. URL http://www.springerlink.com/index/10.1007/BF00992553.

5 Harry Porter Weld. An Experimental Study of Musical Enjoyment. *The American Journal of Psychology*, 23(2): 245–308, 1912. URL http://www.jstor.org/stable/1412844.

6 Klaus R Scherer. What Are Emotions? And How Can They be Measured? *Social Science Information*, 44(4): 695–729, December 2005. ISSN 0539-0184. doi: 10.1177/0539018405058216. URL http://ssi.sagepub.com/cgi/doi/10.1177/0539018405058216.

7 Leonard B Meyer. *Emotion and Meaning in Music*. University of Chicago Press, Chicago, 1956.

8 Klaus R Scherer. Why Music Does not Produce Basic Emotions: Pleading for a New Approach to Measuring the Emotional Effects of Music. In *Proceedings of the Stockholm Music Acoustics Conference (SMAC)*, Stockholm, August 2003.

9 Klaus R Scherer. Which Emotions Can be Induced by Music? What Are the Underlying Mechanisms? And How Can We Measure Them? *Journal of New Music Research*, 33(3): 239–251, September 2004. ISSN 0929-8215. doi: 10.1080/0929821042000317822. URL http://www.informaworld.com/openurl?genre=article&doi=10.1080/0929821042000317822&magic=crossref||D404A21C5BB053405B1A640AFFD44AE3.

10 Marcel Zentner, Didier Grandjean, and Klaus R Scherer. Emotions Evoked by the Sound of Music: Characterization, Classification, and Measurement. *Emotion*, 8(4):494–521, August 2008. ISSN 1528-3542. doi: 10.1037/1528-3542.8.4.494. URL http://www.ncbi.nlm.nih.gov/pubmed/18729581.

11 Patrik N Juslin. Cue Utilization in Communication of Emotion in Music Performance: Relating Performance to Perception. *Journal of Experimental Psychology*, 26(6): 1797–1813, 2000.

12 Joanna Kantor-Martynuska. Emotion-Relevant Characteristics of Temperament and the Perceived Magnitude of Tempo and Loudness of Music. In *Proceedings of the 9th International Conference on Music Perception and Cognition (ICMPC)*, Bologna, August 2006.

13 John A Sloboda and Andreas C Lehmann. Tracking Performance Correlates of Changes in Perceived Intensity of Emotion During Different Interpretation of a Chopin Piano Prelude. *Music Perception*, 19(1):87–120, 2001.

14 Emery Schubert. Modeling Perceived Emotion With Continuous Musical Features. *Music Perception*, 21(4):561–585, 2004.

15 Renee Timmers, Matija Marolt, Antonio Camurri, and Gualtiero Volpe. Listeners' Emotional Engagement with Performances of a Scriabin Étude: An Explorative Case Study. *Psychology of Music*, 34(4):481–510, 2006.

16 Gabriela Husain, William Forde Thompson, and E Glenn Schellenberg. Effects of Musical Tempo and Mode on Arousal, Mood, and Spatial Abilities. *Music Perception*, 20(2):151–171, 2002.

17 Alexander Lerch, Claire Arthur, K Ashis Pati, and Siddharth Gururani. An Interdisciplinary Review of Music Performance Analysis. *Transactions of the International Society for Music Information Retrieval (TISMIR)*, 3(1):221–245, 2020. doi: 10.5334/tismir.53. URL https://transactions.ismir.net/articles/10.5334/tismir.53.

18 Roberto Dillon. Extracting Audio Cues in Real Time to Understand Musical Expressiveness. In *Proceedings of the MOSART workshop*, Barcelona, November 2001.

19 Roberto Dillon. A Statistical Approach to Expressive Intention Recognition in Violin Performances. In *Proceedings of the Stockholm Music Acoustics Conference (SMAC)*, Stockholm, August 2003.

20 Emery Schubert. Update of the Hevner Adjective Checklist. *Perceptual and Motor Skills*, 96:1117–1122, 2003.

21 Xiao Hu and J Stephen Downie. Exploring Mood Metadata: Relationships with Genre, Artist and Usage Metadata. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Vienna, 2007. URL http://ismir2007.ismir.net/proceedings/ISMIR2007_p067_hu.pdf.

22 James A Russell. A Circumplex Model of Affect. *Journal of Personality and Social Psychology*, 39 (6):1161–1178, 1980. ISSN 1939-1315(Electronic);0022-3514(Print). doi: 10.1037/h0077714.

23 David Watson and Auke Tellegen. Toward a Consensual Structure of Mood. *Psychological Bulletin*, 98(2):219–235, 1985. ISSN 1939-1455(Electronic), 0033-2909(Print). doi: 10.1037/0033-2909.98.2.219. Place: US Publisher: American Psychological Association.

24 Robert E Thayer. *The Biopsychology of Mood and Arousal*. Oxford University Press, 1989. ISBN 978-0-19-536175-9.

25 Lisa Feldman Barrett and James A Russell. Independence and Bipolarity in the Structure of Current Affect. *Journal of Personality and Social Psychology*, 74 (4):967–984, 1998. URL https://psycnet.apa.org/doiLanding?doi=10.1037%2F0022-3514.74.4.967.

26 Michelle S M Yik, James A Russell, and Lisa Feldman Barrett. Structure of Self-Reported Current Affect: Integration and Beyond. *Journal of Personality and Social Psychology*, 77 (3):600–619, 1999.

27 Lisa Feldman Barrett and James A Russell. The Structure of Current Affect: Controversies and Emerging Consensus. *Current Directions in Psychological Science*, 8 (1):10–14, February 1999. ISSN 0963-7214, 1467-8721. doi: 10.1111/1467-8721.00003. URL http://journals.sagepub.com/doi/10.1111/1467-8721.00003.

28 Marc Leman, Valery Vermeulen, Liesbeth De Voogdt, Dirk Moelants, and Micheline Lesaffre. Prediction of Musical Affect Using a Combination of Acoustic Structural Cues. *Journal of New Music Research*, 34(1): 39–67, March 2005. ISSN 0929-8215. doi: 10.1080/09298210500123978. URL http://www.tandfonline.com/doi/abs/10.1080/09298210500123978.

29 Karl F MacDorman, Stuart Ough, and Ho Chin-Chang. Automatic Emotion Prediction of Song Excerpts: Index Construction, Algorithm Design, and Empirical Comparison. *Journal of New Music Research*, 36(4): 281–299, December 2007. ISSN 0929-8215. doi: 10.1080/09298210801927846. URL http://www.tandfonline.com/doi/abs/10.1080/09298210801927846.

30 Luca Mion and Giovanni De Poli. Score-Independent Audio Features for Description of Music Expression. *Transactions on Audio, Speech, and Language Processing*, 16(2):458–466, February 2008. ISSN 1558-7916. doi: 10.1109/TASL.2007.913743. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4432649.

31 Yazhong Feng, Yueting Zhuang, and Yunhe Pan. Music Information Retrieval by Detecting Mood via Computational Media Aesthetics. In *Proceedings of the International Conference on Web Intelligence (WI)*, pages 235–241. IEEE, 2003. ISBN 0-7695-1932-6. doi: 10.1109/WI.2003.1241199. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1241199.

32 Lie Lu, Dan Liu, and Hong-Jiang Zhang. Automatic Mood Detection and Tracking of Music Audio Signals. *Transactions on Audio, Speech and Language Processing*, 14(1):5–18, January 2006. ISSN 1558-7916. doi: 10.1109/TSA.2005.860344. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1561259.

33 Yi-Hsuan Yang, Ya-fan Su, Yu-ching Lin, and Homer H Chen. Music Emotion Recognition: The Role of Individuality. In *Proceedings of the International Workshop on Human-centered Multimedia (HCM)*, Augsburg, 2007. ACM. ISBN 978-1-59593-781-0. doi: 10.1145/1290128.1290132.

34 Pasi Saari, Tuomas Eerola, and Olivier Lartillot. Generalizability and Simplicity as Criteria in Feature Selection: Application to Mood Classification in Music. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(6):1802–1812, August 2011. ISSN 1558-7924. doi: 10.1109/TASL.2010.2101596. Conference Name: IEEE Transactions on Audio, Speech, and Language Processing.

35 Tuomas Eerola and Jonna K Vuoskoski. A Comparison of the Discrete and Dimensional Models of Emotion in Music. *Psychology of Music*, 39(1):18–49, August 2010. ISSN 0305-7356. doi: 10.1177/0305735610362821. URL http://pom.sagepub.com/cgi/doi/10.1177/0305735610362821.

36 Tao Li and Mitsunori Ogihara. Content-based Music Similarity Search and Emotion Detection. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 5, pages 705–708. IEEE, 2004. ISBN 0-7803-8484-9. doi: 10.1109/ICASSP.2004.1327208. URL http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1327208.

37 Yi-Hsuan Yang, Yu-Ching Lin, Ya-Fan Su, and Homer H Chen. A Regression Approach to Music Emotion Recognition. *Transactions on Audio, Speech, and Language Processing*, 16(2):448–457, February 2008. ISSN 1558-7916. doi: 10.1109/TASL.2007.911513. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4432654.

38 Arefin Huq, Juan Pablo Bello, and Robert Rowe. Automated Music Emotion Recognition: A Systematic Evaluation. *Journal of New Music Research*, 39(3):227–244, September 2010. ISSN 0929-8215. doi: 10.1080/09298215.2010.513733. URL http://www.tandfonline.com/doi/abs/10.1080/09298215.2010.513733.

39 Yi-Hsuan Yang, Chia-Chu Liu, and Homer H Chen. Music Emotion Classification: a Fuzzy Approach. In *Proceedings of the 14th Annual ACM International*

Conference on Multimedia. ACM, 2006. ISBN 1-59593-447-2. URL http://portal.acm.org/citation.cfm?id=1180665.

40 Konstantinos Trohidis, Grigorios Tsoumakas, George Kalliris, and Ioannis Vlahavas. Multi-Label Classification of Music into Emotions. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Philadelphia, PA, 2008. URL http://books.google.com/books?hl=en&amp;lr=&amp;id=OHp3sRnZD-oC&amp;oi=fnd&amp;pg=PA325&amp;dq=MULTI-LABEL+CLASSIFICATION+OF+MUSIC+INTO+EMOTIONS&amp;ots=oDKOtJdDg2&amp;sig=HgrLzPqJJn64KwCpuRTLLpZ2Dy0.

41 Byeong-jun Han, Seungmin Rho, Sanghoon Jun, and Eenjun Hwang. Music Emotion Classification and Context-Based Music Recommendation. *Multimedia Tools and Applications*, 47(3): 433–460, August 2010. ISSN 1380-7501. doi: 10.1007/s11042-009-0332-6. URL http://www.springerlink.com/index/10.1007/s11042-009-0332-6.

42 Yu-An Chen, Yi-Hsuan Yang, Ju-Chiang Wang, and Homer Chen. The AMG1608 Dataset for Music Emotion Recognition. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 693–697, April 2015. doi: 10.1109/ICASSP.2015.7178058. ISSN: 2379-190X.

43 Anna Aljanaki, Yi-Hsuan Yang, and Mohammad Soleymani. Developing a Benchmark for Emotional Analysis of Music. *PLoS ONE*, 12(3):e0173392, March 2017. ISSN 1932-6203. doi: 10.1371/journal.pone.0173392. URL https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0173392. Publisher: Public Library of Science.

44 Mohammad Soleymani, Micheal N Caro, Erik M Schmidt, Cheng-Ya Sha, and Yi-Hsuan Yang. 1000 Songs for Emotional Analysis of Music. In *Proceedings of the 2nd ACM international workshop on Crowdsourcing for multimedia - CrowdMM '13*, pages 1–6, Barcelona, Spain, 2013. ACM Press. ISBN 978-1-4503-2396-3. doi: 10.1145/2506364.2506365. URL http://dl.acm.org/citation.cfm?doid=2506364.2506365.

45 Renato Panda, Ricardo Malheiro, and Rui Pedro Paiva. Novel Audio Features for Music Emotion Recognition. *IEEE Transactions on Affective Computing*, 11 (4):614–626, October 2020. ISSN 1949-3045. doi: 10.1109/TAFFC.2018.2820691. Conference Name: IEEE Transactions on Affective Computing.

46 Eunjeong Koh and Shlomo Dubnov. Comparison and Analysis of Deep Audio Embeddings for Music Emotion Recognition. In *AAAI Workshop on Affective Content Analysis*, April 2021. URL http://arxiv.org/abs/2104.06517.

14

Musical Instrument Recognition

The target of *musical instrument recognition* systems is the identification of musical instruments present in a recording. There exist a variety of subtasks with different constraints and expectations that can all be summarized as instrument recognition. Generally speaking, these subtasks can be differentiated by input data content and classification target. Constraints on and characteristics of the input data content are, for instance, (i) the length of input signal which can be only a note, a short phrase, or a complete piece of music, and (ii) the number of instruments playing, e.g. one instrument playing a single voiced melody, one instrument playing multiple-voices, or a poly-timbral scenario with multiple instruments playing simultaneously. The classification target might be differentiated by (i) the number of instruments detectable for each input, e.g. identifying only one predominant instrument (possibly in a poly-timbral mixture) or recognizing the presence/absence of all instruments, and (ii) the time resolution of the classification, i.e. detecting the general presence or absence of an instrument in a recording vs. detecting the exact time segments of each instrument's presence.

Musical instrument recognition is, on the one hand, a classification task because it classifies the presence or absence of instruments in a recording. On the other hand, it can be also seen as a transcription task as the definition of instruments and instrumentation is – at least for some genres – a score-inherent property. This is also illustrated by the task of automatic drum transcription (see Section 9.8) which is, on the one hand, detecting the presence of various drum instruments, but, on the other hand, can also be seen as extended onset detection and thus as a transcription task. Singing voice detection, also referred to as vocal activity detection is a special case of musical instrument recognition.

The recognition of musical instruments in audio can be useful for a range of applications. For instance, music recommendation and music discovery systems can utilize this information to match user preferences or allow the user to search for songs with specific instruments only. Instrument recognition can also improve other tasks such as source separation. For instance, Slizovskaia et al.

An Introduction to Audio Content Analysis: Music Information Retrieval Tasks & Applications,
Second Edition. Alexander Lerch.
© 2023 The Institute of Electrical and Electronics Engineers, Inc. Published 2023 by John Wiley & Sons, Inc.

showed that adding instrument activation as an input conditioning could improve results for separating more than two sources [1]. Musical instrument recognition has also been used in a multitask setup to improve music source separation [2].

The possibly wide range of timbre that can be represented by one instrument played with different playing techniques complicates the task. The differences between, for instance, a bowed violin and a *pizzicato* violin are considerable in both the time domain (e.g. attack time) as well as frequency domain (e.g. level of harmonics), while the sound of a viola can, under certain circumstances, be very similar to that of a violin. These challenges resulted in early systems for musical instrument detection targeting the recognition in a monophonic recording, be it either a single note or a melody. Herrera et al. present a survey of the literature on monophonic musical instrument recognition [3]. Many of the early approaches utilize features from the standard feature set introduced in Section 3.5 or similar features [4–6]. Dealing with single-voiced audio, however, enables extending this set with additional features that are not easily extracted from complex polyphonic mixtures, such as *temporal envelope features* describing the temporal evolution of the sound (duration, temporal centroid, logarithmic attack time, etc.) and *pitch-based features* utilizing the fundamental frequency of the sound (energy ratio of even and odd harmonics, the inharmonicity of the harmonics, and the onset asynchrony between harmonics, etc.) [7–13]. The classifiers used reflect the common classifiers at that time: K-nearest Neighbor (KNN), Artificial Neural Network (ANN), Gaussian Mixture Model (GMM), and Support Vector Machine (SVM) are used [14–17].

The detection of the *predominant* instrument in a mixture is considerably more complicated than in a monophonic recording. In this case, only one class has to be detected at each point in time, but the audio signal can contain many different concurrently occurring musical sound sources. A somewhat special case of predominant instrument recognition is the detection of solo instruments in concertos [18]. Early systems use a standard pipeline with feature extraction followed by classification [19] or identify the salient predominant instrument with instrument models based on harmonic sinusoidal peaks [20]. Alternatively, the methods can also be extended by a preprocessing stage separating sources [21]. More modern systems approach this task with Convolutional Neural Networks (CNNs) [22].

The most general task aims at detecting the presence and absence of multiple instruments in a polyphonic mixture. In this case, the sounds of different instruments likely overlap in both time and frequency and are not easily distinguishable and separable for the analysis. An important complication compared to the previously mentioned subtasks is that detecting the presence and absence of multiple instruments for each audio input makes this a multi-label classification problem: an unknown number of instruments can be active at any given time. Some researchers simplify this problem by restricting the number of concurrent instrument to a small number [23–25]. Many approaches combine ideas from source separation to the task of instrument recognition, either by directly integrating source separation as a preprocessing step to feature-based classification [26] or by building source models with sinusoidal analysis [27] or estimating spectral masks [28]. Learned features have also shown promise for the task of music instrument recognition with features learned, for example, through sparse coding [29] or features learned on different tasks [30, 31].

14.1 Evaluation

Since musical instrument recognition is a multi-label task, it is usually evaluated as a problem with multiple binary classifiers. Thus, the reported metrics are usually the average F-measure or the average accuracy.

The biggest challenges are commonly related to annotations and properties of the data themselves. While instrument annotations are less subjective than, for example, genre or mood labels, an annotation is not necessarily without disagreements. Partly, this is because the definition of instrument taxonomies itself can be vague [32] and the hierarchical level in which instruments are annotated is often unclear. For example, while one user annotates *Bass*, another might actually differentiate between *Electric Bass* and *Upright Bass*. One user might annotate *Drums* being present while others might list all individual percussive instruments present. The generation of a taxonomy for classification brings up other fundamental questions: should a taxonomy only take into account the instrument or also the effect, for example, *Electric Guitar* vs. *Acoustic Guitar* or *Distorted Guitar* vs. *Clean Guitar*? While a *Synthesizer* is clearly a musical instrument, it can produce a wide range of different sounds that might imitate other (acoustic) instruments as well – should it still be labeled as synthesizer or as the instrument it is imitating? Looking at symphonic music, should the whole violin section be labeled as *Violin* or *Violin Section*?

The process of manually annotating audio files is a time-consuming task. IRMAS, for instance, a dataset for instrument detection only annotates the predominant instruments in a set of audio recordings [21]. A work-around solution to get full annotations is to automate the process by using multitrack datasets such as MedleyDB [33] or MixingSecrets [34]. Having the individual track per instrument allows to automatically generate frame-level instrument labels based on a simple Root Mean Square (RMS) threshold. As the RMS volume of an instrument does not necessarily equal the likelihood for its presence, the threshold has to be chosen carefully and intelligently. The main advantage of using multitrack data is the automatic generation of fine-grained labels. These data, however, often suffer from a strong imbalance of class labels. Depending on the genre, some instruments are more commonly present than others; a flute, for example is less likely to appear than drums in rock music and will thus have a much smaller amount of annotated data. This is also true within one recording, in which some instruments might be playing nonstop while others are only active in certain parts of the song. User annotation of real-world audio files is – given the necessary amount of data required for training of modern systems – only possible with certain constraints. A good example is the OpenMic dataset [35] which offers annotations for short snippets of audio that are both weak and incomplete. The annotations are referred to as weak as they only indicate the presence or absence for the whole snippet as opposed to a start and end time. The annotations are incomplete as not every audio snippet is labeled with the presence or absence of all instruments; rather, only a few instruments are labeled as present or absent while the status of the remaining instruments remains unknown. These two characteristics lead to several challenges that might require adaptation of the machine learning approach to the data. The weak nature of labels can, for example, be addressed with attention models [36] while semisupervised learning techniques can be applied to utilize the unlabeled parts of the data [37].

Typical overall results often are around an F-measure of 80% for the OpenMic dataset with 20 instrument classes, although the instrument-specific performance can vary from as low as 50% to as high as 95%.

References

1 Olga Slizovskaia, Leo Kim, Gloria Haro, and Emilia Gomez. End-to-end Sound Source Separation Conditioned on Instrument Labels. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 306–310, Brighton, UK, May 2019. doi: 10.1109/ICASSP.2019.8683800. ISSN: 2379-190X.

2 Yun-Ning Hung and Alexander Lerch. Multi-Task Learning for Instrument Activation Aware Music Source Separation. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Montreal, 2020. International Society for Music Information Retrieval (ISMIR). URL https://musicinformatics.gatech.edu/wp-content_nondefault/uploads/2020/08/Hung-and-Lerch-2020-Multi-Task-Learning-for-Instrument-Activation-Awar.pdf.

3 Perfecto Herrera, Xavier Amatriain, Eloi Batlle, and Xavier Serra. Towards Instrument Segmentation for Music Content Description: A Critical Review of Instrument Classification Techniques. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, 2000. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.18.6351&amp;rep=rep1&amp;type=pdf.

4 Judith C Brown. Computer Identification of Musical Instruments Using Pattern Recognition with Cepstral Coefficients as Features. *Journal of the Acoustical Society of America (JASA)*, 105(3):1933–1941, March 1999. ISSN 0001-4966. URL http://www.ncbi.nlm.nih.gov/pubmed/10089614.

5 Janet Marques and Pedro J Moreno. A Study of Musical Instrument Classification Using Gaussian Mixture Models and Support Vector Machines. Technical Report June, Cambridge Research Laboratory CRL 99/4, 1999.

6 Giulio Agostini, Maurizio Longari, and Emanuele Pollastri. Musical Instrument Timbres Classification with Spectral Features. *Journal on Applied Signal Processing*, 1:5–14, 2003.

7 Ian Kaminskyj and A Materka. Automatic Source Identification of Monophonic Musical Instrument Sounds. In *Proceedings of the International Conference on Neural Networks (ICNN)*, volume 1, pages 189–194. IEEE, 1995. ISBN 0-7803-2768-3. doi: 10.1109/ICNN.1995.488091. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=488091.

8 Ian Kaminskyj and P Voumard. Enhanced Automatic Source Identification of Monophonic Musical Instrument Sounds. In *Proceedings of the Australian New Zealand Conference on Intelligent Information Systems (ANZIIS)*, pages 76–79, Adelaide, November 1996. IEEE. ISBN 0-7803-3667-4. doi: 10.1109/ANZIIS.1996.573893. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=573893.

9 Antti Eronen and Anssi P Klapuri. Musical Instrument Recognition using Cepstral Coefficients and Temporal Features. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 2, pages II753–II756. IEEE, 2000. ISBN 0-7803-6293-4. doi: 10.1109/ICASSP.2000.859069. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=859069.

10 Bozena Kostek and Andrzej Czyzewski. Representing Musical Instrument Sounds for their Automatic Classification. *Journal of the Audio Engineering Society (JAES)*, 49(9):768–785, 2001.

11 Antti Eronen. Comparison of Features for Musical Instrument Recognition. In *Proceedings of the Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, New Paltz, 2001.

12 Judith C Brown, Olivier Houix, and Stephen McAdams. Feature Dependence in the Automatic Identification of Musical Woodwind Instruments. *The Journal of the Acoustical Society of America*, 109(3):1064–1072, 2001. ISSN 00014966. doi: 10.1121/1.1342075. URL http://link.aip.org/link/JASMAN/v109/i3/p1064/s1&Agg=doi.

13 Jeremiah Da Deng, Christian Simmermacher, and Stephen Cranefield. A Study on Feature Analysis for Musical Instrument Classification. *Transactions on Systems, Man, and Cybernetics*, 38(2):429–438, April 2008. ISSN 1083-4419.

doi: 10.1109/TSMCB.2007.913394. URL http://www.ncbi.nlm.nih.gov/pubmed/18348925.

14 A G Krishna and T V Sreenivas. Music Instrument Recognition: from Isolated Notes to Solo Phrases. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 4, pages iv–265–iv–268. IEEE, 2003. ISBN 0-7803-8484-9. doi: 10.1109/ICASSP.2004.1326814. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1326814.

15 Arie A Livshin and Xavier Rodet. Musical Instrument Identification in Continuous Recordings. In *Proceedings of the 7th International Conference on Digital Audio Effects (DAFX)*, Naples, 2004.

16 Slim Essid, Gaël Richard, and Bertrand David. Musical Instrument Recognition on Solo Performance. In *Proceedings of the European Signal Processing Conference (EUSIPCO)*, 2004. URL http://www.tsi.enst.fr/ grichard/Publications/Eusipco04.pdf.

17 Slim Essid, Gaël Richard, and Bertrand David. Musical Instrument Recognition by Pairwise Classification Strategies. *IEEE Transactions on Audio, Speech and Language Processing*, 14(4):1401–1412, 2006. ISSN 1558-7916. doi: 10.1109/TSA.2005.860842. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1643665 http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1643665.

18 Jana Eggink and Guy J Brown. Instrument Recognition in Accompanied Sonatas and Concertos. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 4, pages iv–217–iv–220. IEEE, 2004. ISBN 0-7803-8484-9. doi: 10.1109/ICASSP.2004.1326802. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1326802.

19 Ferdinand Fuhrmann, Martín Haro, and Perfecto Herrera. Scalability, Generality and Temporal Aspects in Automatic Recognition of Predominant Musical Instruments in Polyphonic Music. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Kobe, Japan, 2009. URL http://mtg.upf.edu/files/publications/fuhrmann_haro_herrera_ismir09.pdf.

20 Gunnar Eisenberg. *Identifikation und Klassifikation von Musikinstrumentenklängen in monophoner und polyphoner Musik*. Cuvillier Verlag, Göttingen, 2008. ISBN 978-3-86727-825-6.

21 Juan J Bosch, Jordi Janer, Ferdinand Fuhrmann, and Perfecto Herrera. A Comparison of Sound Segregation Techniques for Predominant Instrument Recognition in Musical Audio Signals. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 6, Porto, Portugal, 2012.

22 Yoonchang Han, Jaehun Kim, and Kyogu Lee. Deep Convolutional Neural Networks for Predominant Instrument Recognition in Polyphonic Music.

IEEE/ACM Transactions on Audio, Speech, and Language Processing, 25(1):208–221, 2016. ISSN 2329-9304. doi: 10.1109/TASLP.2016.2632307.

23 Jana Eggink and Guy Brown. A Missing Feature Approach to Instrument Identification in Polyphonic Music. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 5, pages V–553–6. IEEE, 2003. ISBN 0-7803-7663-3. doi: 10.1109/ICASSP.2003.1200029. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1200029.

24 Jana Eggink and Guy J Brown. Application of Missing Feature Theory to the Recognition of Musical Instruments in Polyphonic Audio. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, 2003. 00053.

25 Tetsuro Kitahara, Masataka Goto, Kazunori Komatani, Tetsuya Ogata, and Hiroshi G Okuno. Instrument Identification in Polyphonic Music: Feature Weighting to Minimize Influence of Sound Overlaps. *EURASIP Journal on Applied Signal Processing*, 2007, 2007. doi: 10.1155/2007/51979. URL https://ui.adsabs.harvard.edu/abs/2006EJASP2007..299K. ADS Bibcode: 2006EJASP2007..299K.

26 Toni Heittola, Anssi P Klapuri, and Tuomas Virtanen. Musical Instrument Recognition in Polyphonic Audio Using Source-Filter Model for Sound Separation. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Kobe, 2009.

27 Luis Gustavo Martins, Juan José Burred, George Tzanetakis, and Mathieu Lagrange. Polyphonic Instrument Recognition using Spectral Clustering. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Vienna, 2007.

28 Dimitrios Giannoulis and Anssi Klapuri. Musical Instrument Recognition in Polyphonic Audio Using Missing Feature Approach. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(9):1805–1817, September 2013. ISSN 1558-7916, 1558-7924. doi: 10.1109/TASL.2013.2248720. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6470658.

29 Li-Fan Yu, Li Su, and Yi-Hsuan Yang. Sparse Cepstral Codes and Power Scale for Instrument Identification. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7460–7464, May 2014. doi: 10.1109/ICASSP.2014.6855050. ISSN: 2379-190X.

30 Siddharth Gururani, Cameron Summers, and Alexander Lerch. Instrument Activity Detection in Polyphonic Music using Deep Neural Networks. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Paris, 2018. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2018/06/Gururani-et-al.-Instrument-Activity-Detection-in-Polyphonic-Music-.pdf.

31 Arun Solanki and Sachin Pandey. Music Instrument Recognition Using Deep Convolutional Neural Networks. *International Journal of Information Technology*, January 2019. ISSN 2511-2112. doi: 10.1007/s41870-019-00285-y. URL https://doi.org/10.1007/s41870-019-00285-y.

32 Margaret J Kartomi. *On Concepts and Classifications of Musical Instruments*. University Of Chicago Press, 1990. ISBN 978-0-226-42549-8.

33 Rachel Bittner, Justin Salamon, Mike Tierney, Matthias Mauch, Chris Cannam, and Juan Bello. MedleyDB: A Multitrack Dataset for Annotation-Intensive MIR Research. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 7, Taipei, Taiwan, 2014.

34 Siddharth Gururani and Alexander Lerch. Mixing Secrets: A Multitrack Dataset for Instrument Detection in Polyphonic Music. In *Late Breaking Demo (Extended Abstract), Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Suzhou, 2017. International Society for Music Information Retrieval (ISMIR). URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2017/10/Gururani_Lerch_2017_Mixing-Secrets.pdf.

35 Eric J Humphrey, Simon Durand, and Brian McFee. OpenMIC-2018: An Open Dataset for Multiple Instrument Recognition. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 7, Paris, France, 2018.

36 Siddharth Gururani, Mohit Sharma, and Alexander Lerch. An Attention Mechanism for Music Instrument Recognition. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Delft, 2019. International Society for Music Information Retrieval (ISMIR). URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2019/07/Gururani-et-al.-2019-An-Attention-Mechanism-for-Music-Instrument-Recogn.pdf.

37 Siddharth Gururani and Alexander Lerch. Semi-Supervised Audio Classification with Partially Labeled Data. In *Proceedings of the IEEE International Symposium on Multimedia (ISM)*, online, 2021. Institute of Electrical and Electronics Engineers (IEEE). URL https://arxiv.org/abs/2111.12761.

15

Music Performance Assessment

Many of the audio content analysis (ACA) tasks introduced above focus on transcribing the musical audio signal into a score-like representation. Music performance analysis and its important subtask *music performance assessment* instead focus on the musical performance and its parameters. Typical research questions target, for example, the differences between two performances of the same piece of music, the identification of the most impactful performance parameters, and the effect of specific performance parameter variations on the listener.

Beyond the musicological and psychological interest in answering these questions, music performance assessment targets a set of applications for music instrument education: the ability to automatically assess a student's performance, determine its quality, and understand which characteristics are impacting an assessment enables artificially intelligent software tutors. Such tutoring systems have the potential to transform instrument practice and didactics of music education through low cost, individualized adaptive training sessions.

15.1 Music Performance

Music is a performing art. A composer cannot "communicate" directly with a listener but requires a performer or group of performers who, in a traditional setting, "self-consciously enacts music for an audience" [1]. The composition is a musical "blueprint" that is rendered by the performers into the acoustic realization [2, 3]. This acoustic signal is then received by the listeners. This process can be represented by the chain of music communication as shown in Figure 15.1 [4, 5].

The *composition*, referred to above as musical blueprint can, for example, be represented by a partitura in the case of Western classical music, a lead sheet for jazz, or some other genre-dependent representation describing the compositional content of a piece of music. *Scores* of the Western tradition always contain information on pitch and (relative) duration of each note and often general instructions on

An Introduction to Audio Content Analysis: Music Information Retrieval Tasks & Applications,
Second Edition. Alexander Lerch.
© 2023 The Institute of Electrical and Electronics Engineers, Inc. Published 2023 by John Wiley & Sons, Inc.

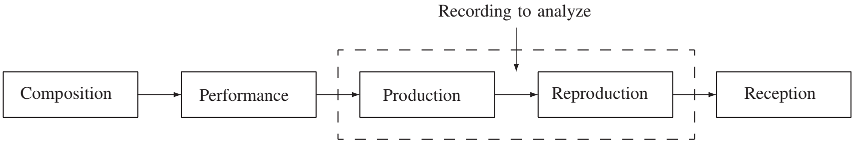


Figure 15.1 Chain of musical communication.

musical dynamics (compare Sections 8.2, 7.2, and 9.2). Additional instructions, for instance, on character, quality, or specific ways to perform may also be found in the score. Some of the contained information is available only implicitly (for example, information on the musical structure) or might be ambiguous or hidden, complicating its description and quantification as pointed out by many musicologists and music psychologists [6–9]. It can be observed that modern scores tend to be more explicit in terms of performance instructions than historic scores, indicating the composers' intention to eliminate the unspecified or ambiguous information in the score [6]. This may be due to the increasing awareness of the fact that scores often take into account performance rules that may seem "natural" at the time of composition but may change over decades and centuries, possibly leading to performance characteristics "unintended" by the composer.

The compositional information is then used by the performers for an acoustic realization of the score, the *music performance*. All information is subject to the performers' interpretation – they detect and evaluate implicit information, try to understand and explain performance instructions, and identify ways to convey their understanding of musical ideas to the listener. By doing so, the performers create (either intentionally or unintentionally, either in preparation or on-the-fly) a performance plan for which the score information might be interpreted, modified, added to, and dismissed [10, 11]. The decisions for the performance plan are based, for example, on (i) general interpretative rules describing norms which performances commonly follow to avoid being perceived as uncommon or unnatural, (ii) the expressive strategy, covering the intended communication of musical structure, the addition of unexpectedness, specific stylistic and cultural rules, and the intended conveyance of mood, (iii) the performers' personal, social, and cultural background, (iv) physical influences such as the physical and cognitive abilities of the performer, (v) the rehearsal experience, and (vi) immediate influences such as the performers' emotional state, room acoustics, and instrument feedback. For more details, see [1, 3, 6, 8, 10, 12–19].

In a live performance setting, no recording, production, or acoustic reproduction takes place and the acoustic rendition is exactly what the listeners will hear depending on the acoustic properties of the venue and their location. The objective quantitative analysis of a performance, however, has to be based on a recording of a signal. Recorded performances differ significantly from the live performance, even in the case of so-called “live recordings” [15, 20]. Mechanical and technological restrictions enforce differences between an original and reproduced performance. Furthermore, choices and interventions by the production team will have impact on the recording. Maempel discusses as main influences in the context of classical music the sound engineer (dynamics, timbre, panorama, depth) and the editor (splicing of different tracks, tempo and timing, pitch) [21]. A performance (plan) can also evolve during a recording session. Katz points out that in such a session, performers will listen to the recording of themselves and adjust “aspects of style and interpretation” [22]. All these manipulations and influences can be part of the audio recording to be analyzed and cannot be separated retroactively from the performers’ creation. As the release of a recording usually has to be preapproved by the performers, it is assumed that the performers’ intent has not been distorted.

The *listener* ultimately hears and processes the performance. Only by asking listeners, for example, can we asses whether the performers’ performance plan was successfully implemented or not and what the aesthetic response of the listener is. Mapping objective performance characteristics to listening responses remains an active area of research that could inform performance models as well as enable a deeper understanding of important musical parameters.

As hinted at above, the chain of musical communication as pictured in Figure 15.1 is a simplified visualization. All stages between performance and listener may feedback information to the performer and subsequently reshape the performance plan.

15.2 Music Performance Analysis

Music Performance Analysis (MPA) focuses on the observation, extraction, description, interpretation, and modeling of music performance parameters as well as the analysis of attributes and characteristics of the generation and perception of music performance. Formally, performance parameters to be analyzed can be grouped into the same basic categories that are used to describe musical audio in general: tempo and timing, dynamics, pitch, and timbre (see Section 2.1). Although the change of performance parameters can have a major impact on a listener’s perception of the music [15], the nature of the performance parameter variation is often subtle, and must be evaluated in reference to something; typically either a recording of the same performer and piece at a different time, different performances of the same piece, or deviations from a perfectly quantized performance rendered from a symbolic score representation (e.g. Musical Instrument Digital Interface (MIDI)) without tempo or dynamics variations. Such deviations and variations have often been described as representing musical expressivity, even if certainly not all deviations can be considered as expressive [23].

As Clarke points out, “musical analysis is not an exact science and cannot be relied upon to provide an unequivocal basis for distinguishing between errors and intentions” [24], emphasizing the challenge of meaningful interpretation of extracted performance data. A related difficulty that MPA has to deal with is the distinction between inherent performance attributes and individual performance attributes. In the context of musical accents, for example, Parncutt [16] distinguishes between *immanent accents* which are assumed to be apparent from the score (structural, harmonic, melodic, metrical, dynamic, instrumental) and *performed accents* “added” to the score by the performer.

Close relationships were observed between musical phrase structure and deviations in tempo and timing [8, 26, 27] and dynamics [19, 28]. For example, tempo changes in the form of *ritardandi* tend to occur at phrase boundaries.

However, simply investigating deviations does not necessarily give an understanding of music performance, as only the listener is the ultimately judge who ultimately consumes, processes, interprets, and assesses the music. The investigation of the perceptual and subjective meaning of various performance characteristics has often resulted in conflicting insights. For example, on the one hand, musical tension has been reported unaffected by flat dynamics, constant tempo or both [29, 30], on the other hand, expressive timing has been found as the most important parameter to convey musical tension [31].

The performance data for MPA is either acquired through sensor-equipped instruments (e.g. Yamaha Disklavier), manual annotation (e.g. annotating onset times in a wave editor software), or by extracting parameters directly from the audio signal with methods described in the previous chapters. All of these different data sources have advantages and disadvantages. Sensors can give very accurate information, however, the use of sensors excludes the analysis of materials recorded outside of the lab setting. Manual annotation is perceptually meaningful but time-consuming and the annotation process is typically regarded as tedious, thus not an option for large amounts of data. Audio content analysis approaches allow access to an enormous and continuously growing heritage of recordings, including outstanding and legendary performances recorded throughout the last century and until now, but it is questionable whether state-of-the-art algorithms can provide the detail and accuracy required for many MPA tasks [25].

15.3 Approaches to Music Performance Assessment

Instead of simply extracting and modeling individual performance parameters, the goal of *music performance assessment* is the estimation of overall ratings for a performance, taking into account parameters spanning the domains of pitch (vibrato rates, intonation, tuning, and temperament), timing (tempo, groove, and microtiming), dynamics (accents, tension), and timbre (playing techniques, instrument settings). This is a task of considerable commercial interest as it enables musically intelligent tutoring software. An automatic system for assessment of performances would provide objective, reliable, and repeatable feedback to the student during practice sessions and increase accessibility of affordable instrument education.

Performance assessment is a ubiquitous aspect of music pedagogy: regular feedback from teachers improves the students' playing and auditions are used to place students in ensembles. It is, however, seen as a highly subjective and aesthetically challenging task with considerable disagreement on assessments between educators [32, 33]. With increasing musical proficiency, the assessment tends to become increasingly influenced by aesthetic considerations, and "technical" skills such as the control of pitch intonation, musical dynamics, and tempo become a minimum requirement. Furthermore, it is important to note that there exists no true, absolute, or optimal interpretation; music is a living art and constant re-interpretation is the artistic breath giving music life [19].

Generally, the structure of a performance assessment system resembles the basic structure of an audio content analysis system: features describing the performance are extracted and then used in the inference stage to estimate one or more ratings. Current systems vary mostly with respect to the features used to describe the music performance, the need for a symbolic reference such as the musical score, the type and granularity of the estimated assessment labels, and the musical instrument they focus on assessing.

While standard instantaneous features as introduced in previous chapters have been used for music performance assessment [34], the majority of systems utilize hand-designed features informed by expert knowledge. Of the four assessment categories introduced above, intonation and timing are most frequently looked at in the literature. Especially features related to pitch deviation and pitch stability [35–38] and tempo and timing stability [39, 40] are frequently used. The analysis of tone quality in music performances is much less common [34, 41, 42]. A score-like reference is commonly available in the educational setting, so many researchers incorporate information from the musical score into feature computation [38, 43–46]. In many cases, such score information is simply taken into account by computing deviations from a "robotic" performance of the score.

The assessment inference can be seen as a classification problem if performances are simply categorized as "good" or "bad" [34, 35], but the prediction via a regression of a rating on a scale, possibly for multiple different performance criteria, is more common [38, 47, 48].

The general trend from feature design toward feature learning can be observed for music performance assessment, albeit at a slower pace than in other fields. Besides the general lack of training data, one of the possible reasons for this reluctance is that an educational setting normally requires not only an accurate assessment but also an explanation of the reasons for that assessment (and possibly a strategy to improve the performance). Many neural network architectures, however, do not allow simple interpretation of the learned features necessary for assessment [49]. Furthermore, most existing neural models for performance assessment estimate a single holistic rating for the whole performance and thus do not allow to pin-point specific playing mistakes.

The success rate of tools for modern automatic performance assessment still leaves considerable room for improvement. Most of the presented systems either work well only for very select data [34] or have comparably low prediction accuracies [37, 38], rendering them unusable in most practical teaching scenarios.

References

1 John A Sloboda. *The Musical Mind – The Cognitive Psychology of Music*. Number 5 in Oxford Psychology Series. Oxford University Press, Oxford, 1985.

2 Peter Hill. From Score to Sound. In John Rink, editor, *Musical Performance – A Guide to Understanding*. Cambridge University Press, Cambridge, 2002. ISBN 978-0-521-78862-5.

3 Eric F Clarke. Understanding the Psychology of Performance. In John Rink, editor, *Musical Performance – A Guide to Understanding*. Cambridge University Press, Cambridge, 2002.

4 Roger A Kendall and Edward C Carterette. The Communication of Musical Expression. *Music Perception*, 8(2):129–164, 1990.

5 Alexander Lerch, Claire Arthur, K Ashis Pati, and Siddharth Gururani. An Interdisciplinary Review of Music Performance Analysis. *Transactions of the International Society for Music Information Retrieval (TISMIR)*, 3(1):221–245, 2020. doi: 10.5334/tismir.53. URL https://transactions.ismir.net/articles/10.5334/tismir.53.

6 Frederick Dorian. *The History of Music in Performance – The Art of Musical Interpretation from the Renaissance to Our Day*. W. W. Norton & Company Inc., New York, 1942.

7 Leonard B Meyer. *Emotion and Meaning in Music*. University of Chicago Press, Chicago, 1956.

8 Caroline Palmer. Music Performance. *Annual Review of Psychology*, 48:115–138, 1997.

9 Jan Beran and Guerino Mazzola. Analyzing Musical Structure and Performance – A Statistical Approach. *Statistical Science*, 14(1):47–79, 1999.

10 John A Sloboda. Music Performance. In Diana Deutsch, editor, *The Psychology of Music*. Academic Press, New York, 1982.

11 Alf Gabrielsson. The Performance of Music. In Diana Deutsch, editor, *The Psychology of Music*. Academic Press, San Diego, CA, 2nd edition, 1999.

12 Neil P McAngus Todd. Vestibular Feedback in Musical Performance. *Music Perception*, 10(3):379–382, 1993.

13 Renee Timmers and Henkjan Honing. On Music Performance, Theories, Measurement and Diversity. *Cognitive Processing*, 1–2: 1–19, 2002.

14 Peter Walls. Historical Performance and the Modern Performer. In John Rink, editor, *Musical Performance – A Guide to Understanding*. Cambridge University Press, Cambridge, 2002.

15 Eric F Clarke. Listening to Performance. In John Rink, editor, *Musical Performance – A Guide to Understanding*. Cambridge University Press, Cambridge, 2002.

16 Richard Parncutt. Accents and Expression in Piano Performance. In Klaus W Niemöller, editor, *Perspektiven und Methoden einer Systemischen Musikwissenschaft*. Peter Lang, Frankfurt/Main, 2003.

17 Patrik N Juslin. Five Myths about Expressivity in Music Performance and What to Do About Them. In *Proceedings of the International Conference on Arts and Humanities*, Honululu, Hawaii, 2003.

18 Patrik N Juslin. Studies of Music Performance: A Theoretical Analysis of Empirical Findings. In *Proceedings of the Stockholm Music Acoustics Conference (SMAC)*, Stockholm, August 2003.

19 Alexander Lerch. *Software-Based Extraction of Objective Parameters from Music Performances*. GRIN Verlag, München, 2009. ISBN 978-3-640-29496-1. URL 10.14279/depositonce-2025.

20 Peter Johnson. The Legacy of Recordings. In John Rink, editor, *Musical Performance – A Guide to Understanding*. Cambridge University Press, Cambridge, 2002.

21 Hans-Joachim Maempel. Musikaufnahmen als Datenquellen der Interpretationsanalyse. In Heinz von Lösch and Stefan Weinzierl, editors, *Gemessene Interpretation - Computergestützte Aufführungsanalyse im Kreuzverhör der Disziplinen*, Klang und Begriff, pages 157–171. Schott, Mainz, 2011. ISBN 978-3-7957-0771-2.

22 Mark Katz. *Capturing Sound: How Technology has Changed Music*. University of California Press, Berkeley and Los Angeles, 2004. ISBN 0-520-24380-3.

23 Johanna Devaney. Inter-versus Intra-Singer Similarity and Variation in Vocal Performances. *Journal of New Music Research*, 45(3):252–264, July 2016. ISSN 0929-8215. doi: 10.1080/09298215.2016.1205631. URL https://doi.org/10.1080/09298215.2016.1205631.

24 Eric F Clarke. Empirical Methods in the Study of Performance. In Eric Clarke and Nicholas Cook, editors, *Empirical Musicology*. Oxford University Press, Oxford, 2004.

25 Alexander Lerch, Claire Arthur, Ashis Pati, and Siddharth Gururani. Music Performance Analysis: A Survey. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Delft, 2019. International Society for Music Information Retrieval (ISMIR). URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2019/06/Lerch-et-al.-2019-Music-Performance-Analysis-A-Survey.pdf.

26 Dirk-Jan Povel. Temporal Structure of Performed Music. Some Preliminary Observations. In *Acta Psychologica*, 41:309–320, 1977.

27 L Henry Shaffer. Timing in Solo and Duet Piano Performances. *The Quarterly Journal of Experimental Psychology*, 36A:577–595, 1984.

28 Bruno H Repp. The Dynamics of Expressive Piano Performance: Schumann's 'Träumerei' Revisited. *Journal of the Acoustical Society of America (JASA)*, 100(1):641–650, 1996.

29 Carol L Krumhansl. A Perceptual Analysis of Mozart's Piano Sonata K. 282: Segmentation, Tension, and Musical Ideas. *Music Perception: An Interdisciplinary Journal*, 13 (3):401–432, April 1996. ISSN 0730-7829, 1533-8312. doi: 10.2307/40286177. URL https://mp.ucpress.edu/content/13/3/401.

30 Morwaread Mary Farbood and Finn Upham. Interpreting Expressive Performance through Listener Judgments of Musical Tension. *Frontiers in Psychology*, 4, 2013. ISSN 1664-1078. doi: 10.3389/fpsyg.2013.00998. URL https://www.frontiersin.org/articles/10.3389/fpsyg.2013.00998/full.

31 Bruno Gingras, Marcus T Pearce, Meghan Goodchild, Roger T Dean, Geraint Wiggins, and Stephen McAdams. Linking Melodic Expectation to Expressive Performance Timing and Perceived Musical Tension. *Journal of Experimental Psychology: Human Perception and Performance*, 42(4):594–609, 2016. ISSN 1939-1277(Electronic), 0096-1523(Print). doi: 10.1037/xhp0000141.

32 Sam Thompson and Aaron Williamon. Evaluating Evaluation: Musical Performance Assessment as a Research Tool. *Music Perception: An Interdisciplinary Journal*, 21(1):21–41, September 2003. ISSN 0730-7829, 1533-8312. doi: 10.1525/mp.2003.21.1.21. URL https://mp.ucpress.edu/content/21/1/21.

33 Brian C Wesolowski, Stefanie A Wind, and George Engelhard. Examining Rater Precision in Music Performance Assessment: An Analysis of Rating Scale Structure Using the Multifaceted Rasch Partial Credit Model. *Music Perception: An Interdisciplinary Journal*, 33(5):662–678, June 2016. ISSN 0730-7829, 1533-8312. doi: 10.1525/mp.2016.33.5.662. URL https://mp.ucpress.edu/content/33/5/662.

34 Trevor Knight, Finn Upham, and Ichiro Fujinaga. The Potential for Automatic Assessment of Trumpet Tone Quality. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 573–578, Miami, 2011.

35 Tomoyasu Nakano, Masataka Goto, and Yuzuru Hiraga. An Automatic Singing Skill Evaluation Method for Unknown Melodies using Pitch Interval Accuracy and Vibrato Features. *Rn*, 12:1, 2006. URL https://staff.aist.go.jp/t.nakano/PAPER/INTERSPEECH2006nakano.pdf.

36 Yin-Jyun Luo. Detection of Common Mistakes in Novice Violin Playing. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 316–322, Malaga, 2015. URL http://ismir2015.uma.es/articles/197_Paper.pdf.

37 Chih-Wei Wu, Siddharth Gururani, Christopher Laguna, Ashis Pati, Amruta Vidwans, and Alexander Lerch. Towards the Objective Assessment of Music Performances. In *Proceedings of the International Conference on Music Perception and Cognition (ICMPC)*, pages 99–103, San Francisco, CA, 2016. ISBN 1-879346-65-5. URL http://www.icmpc.org/icmpc14/proceedings.html.

38 Amruta Vidwans, Siddharth Gururani, Chih-Wei Wu, Vinod Subramanian, Rupak Vignesh Swaminathan, and Alexander Lerch. Objective Descriptors for the Assessment of Student Music Performances. In *Proceedings of the AES Conference on Semantic Audio*, Erlangen, 2017. Audio Engineering Society (AES). URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2017/06/Vidwans-et-al_2017_Objective-descriptors-for-the-assessment-of-student-music-performances.pdf.

39 Jakob Abeßer, Johannes Hasselhorn, Sascha Grollmisch, Christian Dittmar, and Andreas Lehmann. Automatic Competency Assessment of Rhythm Performances of Ninth-grade and Tenth-grade Pupils. In *Proceedings of the International Computer Music Conference (ICMC)*, Athens, 2014.

40 Chih-Wei Wu and Alexander Lerch. Learned Features for the Assessment of Percussive Music Performances. In *Proceedings of the International Conference on Semantic Computing (ICSC)*, Laguna Hills, 2018. IEEE. doi: 10.1109/ICSC.2018.00022. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2018/01/Wu_Lerch_2018_Learned-Features-for-the-Assessment-of-Percussive-Music-Performances.pdf.

41 Oriol Romani Picas, H Parra Rodriguez, Dara Dabiri, Hiroshi Tokuda, Wataru Hariya, Koji Oishi, and Xavier Serra. A Real-Time System for Measuring Sound Goodness in Instrumental Sounds. In *Proceedings of the Audio Engineering Society Convention*, volume 138, Warsaw, 2015.

42 Krish Narang and Preeti Rao. Acoustic Features for Determining Goodness of Tabla Strokes. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 257–263, Suzhou, China, 2017.

43 Oscar Mayor, Jordi Bonada, and Alex Loscos. Performance Analysis and Scoring of the Singing Voice. In *Proceedings of the Audio Engineering Society Convention*, pages 1–7, 2009. URL http://www.researchgate.net/profile/Jordi_Bonada/publication/228755768_Performance:analysis_and_scoring_of_the_singing_voice/links/0912f510beb308775b000000.pdf.

44 Johanna Devaney, Michael I Mandel, Daniel P W Ellis, and Ichiro Fujinaga. Automatically Extracting Performance Data from Recordings of Trained

Singers. *Psychomusicology: Music, Mind and Brain*, 21(1–2):108–136, 2011. ISSN 2162-1535(Electronic), 0275-3987(Print). doi: 10.1037/h0094008.

45 Bar?? Bozkurt, Ozan Baysal, and Deniz Yüret. A Dataset and Baseline System for Singing Voice Assessment. In *Proceedings of the International Symposium on CMMR*, Matosinhos, 2017.

46 Jiawen Huang, Yun-Ning Hung, K Ashis Pati, Siddharth Gururani, and Alexander Lerch. Score-informed Networks for Music Performance Assessment. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Montreal, 2020. International Society for Music Information Retrieval (ISMIR). URL https://musicinformatics.gatech.edu/wp-content_nondefault/uploads/2020/08/Huang-et-al.-2020-Score-informed-Networks-for-Music-Performance-Asse.pdf.

47 Kumar Ashis Pati, Siddharth Gururani, and Alexander Lerch. Assessment of Student Music Performances Using Deep Neural Networks. *Applied Sciences*, 8(4):507, March 2018. doi: 10.3390/app8040507. URL http://www.mdpi.com/2076-3417/8/4/507/pdf.

48 Pavan Seshadri and Alexander Lerch. Improving Music Performance Assessment with Contrastive Learning. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 8, Online, 2021. URL https://arxiv.org/abs/2108.01711.

49 Vsevolod Eremenko, Alia Morsi, Jyoti Narang, and Xavier Serra. Performance Assessment Technologies for the Support of Musical Instrument Learning. In *Proceedings of the International Conference on Computer Supported Education (CSEDU)*, page 12, Prague, 2020.

Part IV

Appendices

Appendix A

Fundamentals

This chapter reintroduces and summarizes some of the important characteristics of digital audio signals and tools used in signal processing and Audio Content Analysis (ACA). Furthermore, it will introduce some definitions used throughout the book. In order to keep this text short, some of the definitions may lack derivation since it is not possible to give an extensive introduction to digital audio signal processing in this context. The interested reader may refer to the books by Oppenheim and Schafer [1], Ohm and Lüke [2], and Smith [3] for more complete introductions.

A.1 Sampling and Quantization

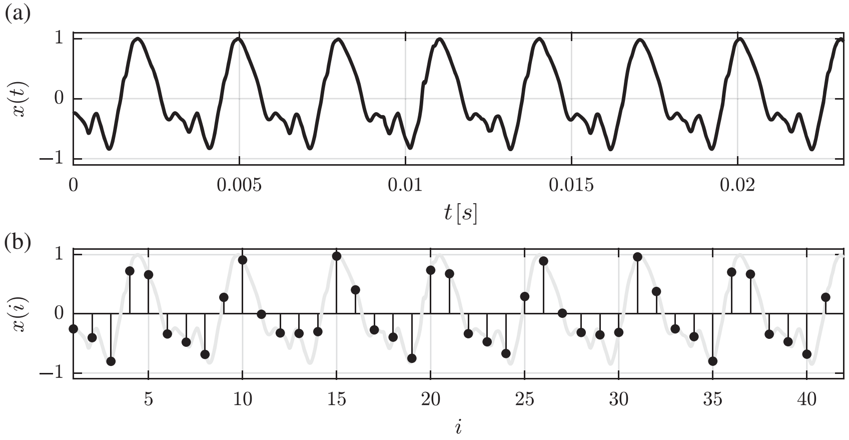
In the digital domain, we cannot deal with continuous signals. Therefore, the analog, continuous signal $x(t)$ has to be discretized in both amplitude and time, where *quantization* refers to the discretization of amplitudes and *sampling* refers to the discretization in time. The resulting signal is a series of quantized amplitude values $x(i)$.

A.1.1 Sampling

The discretization in time is done by *sampling* the signal at specific times. The distance T_S in seconds between sampling points is constant in the context of audio signals and can be derived directly from the *sample rate* f_S by

$$T_\mathrm{S} = \frac{1}{f_\mathrm{S}}. \tag{A.1}$$

An Introduction to Audio Content Analysis: Music Information Retrieval Tasks & Applications,
Second Edition. Alexander Lerch.
© 2023 The Institute of Electrical and Electronics Engineers, Inc. Published 2023 by John Wiley & Sons, Inc.



Repo 1: plotSampling01.m

Figure A.1 Continuous audio signal (a) and corresponding sample values (b) at a sample rate of $f_S = 700$ Hz.

Figure A.1 visualizes the process of sampling by plotting a continuous signal (a) and a sampled representation of it (b) sampled at a sample rate of $f_S = 700$ Hz.

An important property of sampled signals is that the sampled representation is ambiguous as exemplified in Figure A.2. This figure shows that different input signals may lead to the same series of samples.

Repo 1: plotSampling02.m

Figure A.2 Continuous (top) and sampled (below) sinusoidal signals with the frequencies 1 kHz (left), 5 kHz (mid), 7 kHz (right). The sample rate is $f_S = 6$ kHz.

The signal can only be reconstructed unambiguously when certain requirements for the audio signal and sample rates are met as defined by the *sampling theorem*:

Sampling Theorem

A sampled signal can only be reconstructed without loss of information if the sample rate f_S is higher than twice the bandwidth f_max of the audio signal.

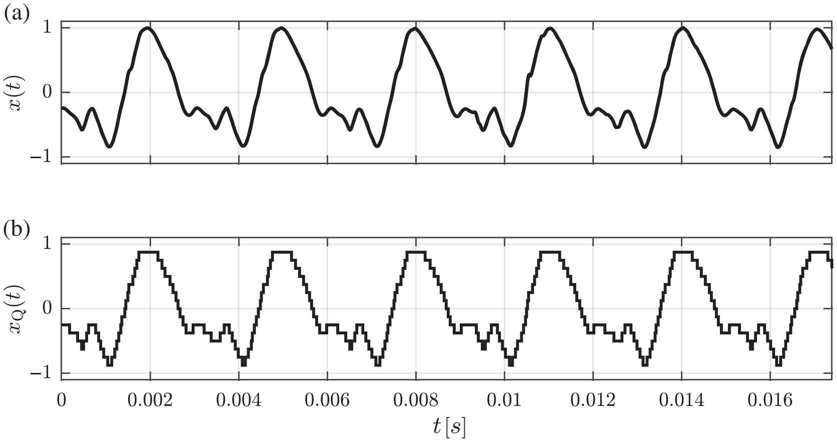
$$f_\mathrm{S} > 2 \cdot f_\mathrm{max} \tag{A.2}$$

Historically, the sampling theorem is attributed to Kotelnikov [4] and Shannon [5] and is based on work by Whittaker [6] and Nyquist [7].

If the signal contains frequency components higher than half the sample rate, these components will be mirrored back and *aliasing* occurs. See also Appendix B.3 for a frequency domain visualization of this artifact.

A.1.2 Quantization

In order to discretize the signal amplitudes, the signal is quantized. *Quantization* means that each amplitude of the signal is rounded to a predefined set of allowed amplitude values. Figure A.3 shows a signal in its continuous and quantized form. Usually, the input amplitude range between the maximum and the minimum amplitude is assumed to be symmetric around 0 and is divided into



Repo 1: plotQuantization.m

Figure A.3 Unquantized input signal (a) and quantized signal at a word length of 4 Bit (b).

$\mathcal{M}$ steps. If $\mathcal{M}$ is a power of 2, the amplitude of each quantized sample can easily be represented with a binary code of *word length*

$$w = \log_2(\mathcal{M}). \tag{A.3}$$

Since $\mathcal{M}$ is an even number, it is impossible to both represent zero amplitude as a quantization step and maintain a symmetric number of quantization steps above and below zero. The usual approach is to have one step less for positive values (see Figure A.4). Typical word lengths for musical audio signals are 16, 24, and 32 Bits. Figure A.4 shows the characteristic line of a quantizer and the corresponding *quantization error* in dependence of the input amplitude.

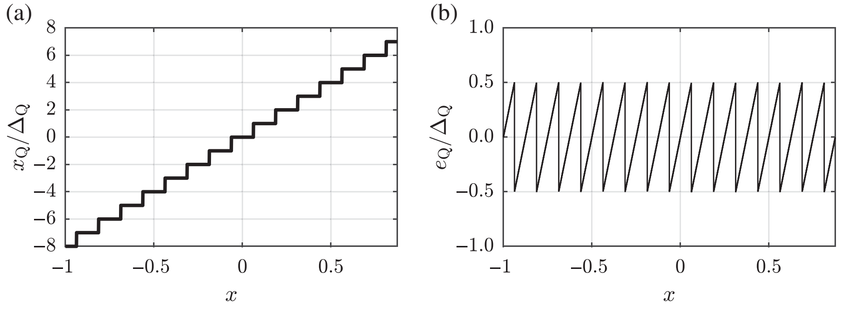
The distance Δ_Q between two neighboring quantization steps is usually constant for all steps. That means that the quantization error e_Q is always in the range $[-\Delta_\mathrm{Q}/2, \Delta_\mathrm{Q}/2]$ if the maximum input signal amplitude is within the range spanned by the maximum allowed quantized amplitudes (no *clipping*). The quantization error is the difference between the original signal x and the quantized signal x_Q:

$$e_\mathrm{Q}(i) = x(i) - x_\mathrm{Q}(i). \tag{A.4}$$

If the input signal amplitudes are much larger than the step size Δ_Q, but in the range of allowed amplitudes, the distribution of quantization error amplitudes can be assumed to be rectangular. The power of the quantization error P_Q can then be estimated with its *PDF* $p_q(e_\mathrm{Q})$ (see Section 3.1.3):

$$P_\mathrm{Q} = \int\limits_{-\infty}^{\infty} e_\mathrm{Q}^2 \cdot p_q(e_\mathrm{Q})\, de_\mathrm{Q} = \frac{1}{\Delta_\mathrm{Q}} \int\limits_{-\Delta_\mathrm{Q}/2}^{\Delta_\mathrm{Q}/2} e_\mathrm{Q}^2\, de_\mathrm{Q} = \frac{\Delta_\mathrm{Q}^2}{12}. \tag{A.5}$$

It is obvious that the power of the quantization error will decrease with decreasing step size Δ_Q and thus with increasing word length w, respectively.



Repo 1: plotQuantizationError.m

Figure A.4 Characteristic line of a quantizer with word length $w = 4$ Bits showing the quantized output amplitude for a normalized input amplitude (a) and the quantization error with respect to the input amplitude (b).

The *signal-to-noise ratio (SNR)* in decibel is a good criterion of the quality of a quantization system. The SNR can be calculated from the ratio of the signal power P_S and the quantization error or noise power P_Q by

$$\mathrm{SNR} = 10\log_{10}\left(\frac{P_\mathrm{S}}{P_\mathrm{Q}}\right)\ [\mathrm{dB}]. \tag{A.6}$$

A high SNR thus indicates good quantization quality. Using Eqs. (A.5) and (A.6) it follows that increasing the word length by one Bit gains approximately 6 dB:

Signal-to-noise ratio (SNR)

$$\mathrm{SNR} = 6.02 \cdot w + c_\mathrm{S}\ [\mathrm{dB}] \tag{A.7}$$

The constant c_S depends on the signal's amplification and PDF:

- square wave (full scale): $c_\mathrm{S} = 10.80$ dB
- sinusoidal wave (full scale): $c_\mathrm{S} = 1.76$ dB
- rectangular PDF (full scale): $c_\mathrm{S} = 0$ dB
- Gaussian PDF (full scale $= 4\sigma_g$)[1]: $c_\mathrm{S} = -7.27$ dB

Assuming that a real-world audio signal is closer to the PDF of noise with Gaussian PDF than to that of a sine wave, the typical SNR is approximately 8 dB lower than commonly stated. Furthermore, the above reference values are given for signals that are leveled more or less optimally; if the input signal is scaled down, the SNR will decrease accordingly (by 6.02 dB per scaling factor ½).

The term *full scale* refers to the highest quantization step. A full-scale sine wave will thus have an amplitude equaling the highest quantization step (see also page 218).

These considerations are often only of limited use to the signal processing algorithm designer, as the quantized signal is usually converted to a signal in floating point format, effectively resulting in a nonlinear characteristic line of the quantizer with a high resolution. The floating point stores the number's mantissa and exponent separately so that the quantization step size Δ_Q increases with the input amplitude. In practice, the signals in floating point format are treated algorithmically as if they had continuous amplitude values, even though the original quantization error still persists. The choice of a maximum amplitude is arbitrary in floating point format; the most common way is to map full scale to the range $[-1, 1]$ to make the signal range independent of the quantizer's word length.

---

[1] Using the approximation that no values are clipped.

A.2 Convolution

Every linear, time-invariant system can be completely described by its *impulse response* $h(i)$, which is the output or the response of a system to an impulse as an input signal. Examples of such systems (or systems which can be approximated as such systems) are filters, speakers, microphones, rooms, etc. The output $y(i)$ of a discrete linear and time-invariant system can be computed from input signal $x(i)$ and the system's impulse response $h(i)$ of length $\mathcal{J}$ by the *convolution* operation:

$$y(i) = x(i) * h(i) = \sum_{j=0}^{\mathcal{J}-1} h(j) \cdot x(i-j). \tag{A.8}$$

The most important properties of the convolution operation are summarized in Eqs. (A.9)–(A.12).

A.2.1 Identity

The result of a convolution with a delta function is the signal itself:

$$x(i) = \delta(i) * x(i).$$

The result for each individual sample can be computed by the sum of the sample itself (weighted by 1) and all other samples weighted by 0, i.e. the sample value itself (compare Eq. (17.39)).

Convolution Properties

- Identity for convolution with the delta function $\delta(i)$:

$$x(i) = x(i) * \delta(i) \tag{A.9}$$

- Commutativity:

$$y(i) = x(i) * h(i) = h(i) * x(i) \tag{A.10}$$

- Associativity:

$$g(i) * h(i) * x(i) = (g(i) * h(i)) * x(i) = g(i) * (h(i) * x(i)) \tag{A.11}$$

- Distributivity:

$$g(i) * (h(i) + x(i)) = (g(i) * h(i)) + (g(i) * x(i)) \tag{A.12}$$

- Circularity: If $h(i)$ is periodic, then the convolution result $y(i) = h(i) * x(i)$ will also be periodic.

A.2.2 Commutativity

Changing the order of operands does not change the result of the convolution operation. That means that the distinction between impulse response and signal is of no mathematical consequence in the context of convolution:

$$h(i) * x(i) = x(i) * h(i).$$

This can be shown by substituting $j' = i - j$:

$$\begin{aligned} x(i) * h(i) &= \sum_{j=-\infty}^{\infty} h(j) \cdot x(i-j) \\ &= \sum_{j'=-\infty}^{\infty} h(i-j') \cdot x(j') \\ &= \sum_{j'=-\infty}^{\infty} x(j') \cdot h(i-j'). \end{aligned} \tag{A.13}$$

A.2.3 Associativity

The associative property of convolution means that changing the order of subsequent convolution operations does not change the overall result. For example, when applying two or more filters to a signal, the output will be identical for every order of filters.[2] This means that

$$(g(i) * h(i)) * x(i) = g(i) * (h(i) * x(i)).$$

This can be derived by changing the order of sums and shifting the operands as shown below:

$$\begin{aligned} (g(i) * h(i)) * x(i) &= \sum_{j=-\infty}^{\infty} (g(j) * h(j)) \cdot x(i-j) \\ &= \sum_{j=-\infty}^{\infty} \left(\sum_{l=-\infty}^{\infty} g(l) \cdot h(j-l) \right) \cdot x(i-j) \\ &= \sum_{j=-\infty}^{\infty} \left(\sum_{l=-\infty}^{\infty} g(l) \cdot h(j-l) \cdot x(i-j) \right) \\ &= \sum_{l=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} g(l) \cdot h(j-l) \cdot x(i-j) \\ &= \sum_{l=-\infty}^{\infty} g(l) \cdot \sum_{j=-\infty}^{\infty} h(j-l) \cdot x(i-j) \end{aligned}$$

---

[2] Strictly speaking this is only true for unlimited word length. The lower the word length, the more the output signal might differ from the expected signal.

$$
\begin{aligned}
&= \sum_{l=-\infty}^{\infty} g(l) \cdot \sum_{j=-\infty}^{\infty} h(j) \cdot x(i-l-j) \\
&= \sum_{l=-\infty}^{\infty} g(l) \cdot (h(i-l) * x(i-l)) \\
&= g(i) * (h(i) * x(i)) \,. \qquad \text{(A.10 revisited)}
\end{aligned}
$$

A.2.4 Distributivity

The order of different linear operations is irrelevant due to the distributive property of the convolution, for example:

$$g(i) * (h(i) + x(i)) = g(i) * h(i) + g(i) * x(i).$$

This means that two signals, one computed by applying a filter to two different signals and summing them together afterward, the other computed by applying the filter to the sum of the signals, are identical:

$$
\begin{aligned}
g(i) * (h(i) + x(i)) &= \sum_{j=-\infty}^{\infty} g(j) \cdot (h(i-j) + x(i-j)) \\
&= \sum_{j=-\infty}^{\infty} g(j) \cdot h(i-j) + g(j) \cdot x(i-j) \\
&= \sum_{j=-\infty}^{\infty} g(j) \cdot h(i-j) + \sum_{j=-\infty}^{\infty} g(j) \cdot x(i-j) \\
&= g(i) * h(i) + g(i) * x(i). \qquad \text{(A.11 revisited)}
\end{aligned}
$$

A.2.5 Circularity

The convolution with a periodic signal will result in a periodic output signal. The periodic signal $x(i)$ is the sum of the shifted (fundamental) periods with length N:

$$x(i) = \sum_{n=-\infty}^{\infty} x_N(i + nN) \tag{A.14}$$

with $x_N(i) = 0$ for $i < 0 \vee i \geq N$. We can show that

$$
\begin{aligned}
x(i) * h(i) &= \sum_{j=-\infty}^{\infty} h(i-j) \sum_{n=-\infty}^{\infty} x_N(j + nN) \\
&= \sum_{n=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} h(i-j) \cdot x_N(j + nN) \\
&= \sum_{n=-\infty}^{\infty} x_N(i + nN) * h(i). \qquad \text{(A.15)}
\end{aligned}
$$

The multiplication of two spectra computed with the Discrete Fourier Transform (DFT) will result in a circular convolution; the result will be the convolution of the two periodically continued sample blocks.

A.2.6 Simple Filter Examples

One typical linear system in digital signal processing is the filter. Every filtering process is a convolution, but for practical filter implementations, this is only of interest for filters with a finite length impulse response, so-called *Finite Impulse Response (FIR)* filters. Filters with an infinite length impulse response are unsurprisingly referred to as *Infinite Impulse Response (IIR)* filters and have a feedback path which complicates the determination of the impulse response in comparison with FIR filters for which the impulse response simply equals its filter coefficients.

In the following, two simple and frequently used digital low-pass filters will be presented, the *Moving Average (MA)* filter as a prototypical FIR filter and the single-pole filter as a simple IIR filter. Both are also often referred to as smoothing filters.

A.2.6.1 Moving Average Filter

The filter equation for an MA filter with an impulse response of length $\mathcal{J}$ is

$$y(i) = \sum_{j=0}^{\mathcal{J}-1} b(j) \cdot x(i-j). \tag{A.16}$$

The coefficients $b(j)$ of a typical MA filter have two main properties; the filter coefficients are identical:

$$b(0) = b(j) \quad \text{for } \; 0 \leq j \leq \mathcal{J} - 1, \tag{A.17}$$

and the sum of all coefficients is normalized to 1:

$$\sum_{j=0}^{\mathcal{J}-1} b(j) = 1. \tag{A.18}$$

Equations (A.17) and (A.18) result in the coefficients $b(j)$ being identical and normalized to $b(j) = {}^{1}\!/_{\mathcal{J}}$. Alternatively, they can also be weighted by an arbitrary window function. Typical window functions are symmetric and weight center samples higher than lateral samples. The window shape allows to elongate the impulse response while reducing the impact of lateral samples on the result.

A simple example for a nonrectangular impulse response shape would be a triangular shape. Applying a MA filter with unweighted coefficients twice to the signal is equivalent to applying a MA filter with a triangular-shaped impulse response of double length. Thus, the repeated use of a MA filter is similar to increasing the impulse response length and applying a window function to the coefficients.

Repo 1: plotLowPass.m

Figure A.5 Magnitude frequency response of a moving average low-pass filter with the impulse response lengths of 2, 10, and 50 samples (a) and frequency response of a single-pole, low-pass filter with an α of 0.5, 0.9, and 0.999 (b).

The MA filter's cut-off frequency, the frequency at which the magnitude first drops by 3 dB, will decrease with an increasing number of coefficients as depicted in Figure A.5 (a). Smith gives a detailed introduction to MA filters in [8].

The number of multiply and add operations per hop increases linearly with the impulse response length of the MA filter. If no window function had been applied to the coefficients, i.e. all coefficients are identical $b(j) = b$, the number of operations can be drastically reduced by implementing the filter recursively:

$$\begin{aligned}
y(i) &= \sum_{j=0}^{\mathcal{J}-1} b \cdot x(i-j) \\
&= b \cdot (x(i) - x(i-\mathcal{J})) + \underbrace{\sum_{j=1}^{\mathcal{J}} b \cdot x(i-j)}_{y(i-1)} \\
&= b \cdot (x(i) - x(i-\mathcal{J})) + y(i-1). \qquad \text{(A.19)}
\end{aligned}$$

This recursive implementation is not applicable to weighted filter coefficients.

As the filter coefficients of the MA filter are symmetric with $h(i) = h(\mathcal{J} - 1 - i)$ (which is only possible for an FIR filter), then the filter will have a linear phase response and thus a constant group delay.

A.2.6.2 Single-Pole Low-Pass Filter

The *single-pole filter* is – mainly due to its simplicity and efficiency – one of the most frequently used low-pass filters for smoothing. The filter equation is

$$y(i) = \alpha \cdot y(i-1) + (1-\alpha) \cdot x(i) \quad \text{with} \ \ 0 \leq \alpha < 1. \qquad \text{(A.20)}$$

Figure A.5b shows the magnitude of the frequency response for three different coefficients α. The coefficient α depends on the required integration time T_{I}; if the integration is defined to be the time required for the filter's response on a step function to rise from 10% to 90%,[3] the coefficient is

$$\alpha = \exp\left(\frac{-2.2}{f_\mathrm{S} \cdot T_\mathrm{I}}\right). \tag{A.21}$$

The cut-off frequency is then

$$\omega_0(\alpha) = \arccos\left(2 - \cosh\left(\log(\alpha)\right)\right) \tag{A.22}$$

which is only defined for $\alpha \geq e^{-\mathrm{arccosh}(3)}$.

A.2.7 Zero-Phase Filtering with IIRs

No nontrivial causal system provides a *zero phase response*. A zero phase response means that the group delay at each frequency is zero as well. Zero-phase response systems output the unmodified timing characteristics of each individual frequency group.

In the case of linear phase FIR filters, a zero-phase response can be reconstructed in an anti-causal way by removing the introduced delay (also referred to as latency) from the filter output (which could also be interpreted as moving the impulse response to be symmetric around time 0).

In the case of IIR filters, however, this procedure is not applicable. In a nonreal-time environment in which the whole input file is available, it is, however, possible to generate a filtered signal with zero-phase shift by using an IIR filter. This can be done by applying the filter twice in series to the signal while using the time-reversed input signal the second time. The second filter run has the effect of leveling out the phase shifts introduced by the first run. The magnitude response of the complete filter process is then the filter's squared magnitude response. Figure A.6 visualizes the intermediate and the final result of this process in the time domain.

To explain this behavior mathematically, the following property is of importance:

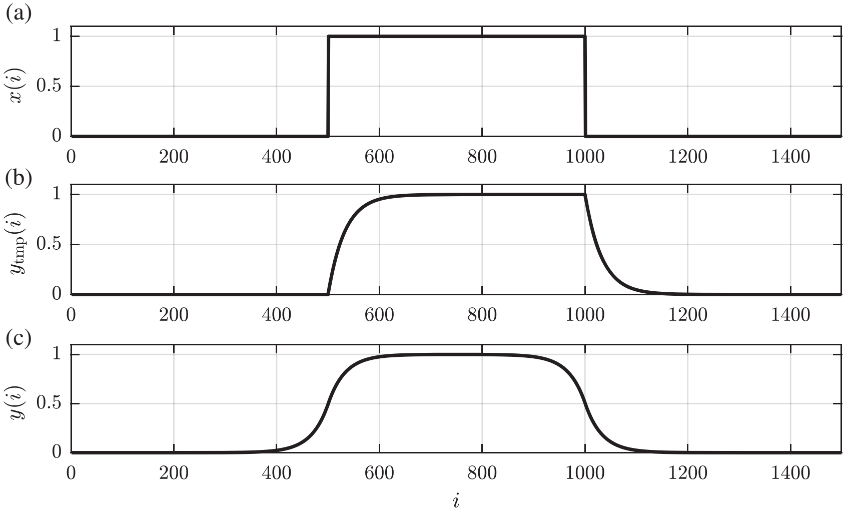
$$\mathfrak{F}\left\{h(-i)\right\} = H\left(-\mathrm{j}\omega\right), \tag{A.23}$$

meaning that the *Fourier transform (FT)* $\mathfrak{F}$ (see Section 3.3.1 and Appendix B) of a time-reversed signal equals the complex-conjugate of the transform of the original signal. The flip function is a tool which helps us in the derivation:

$$\mathrm{flip}\left(x(i)\right) = x(-i). \tag{A.24}$$

---

[3] Note that there exist other approaches to defining the integration time that may differ significantly.



Repo 1: plotZeroPhase.m

Figure A.6 Example of zero phase filtering: input signal (a), low-pass filtered signal after the first forward pass (b), and final result of both forward and backward paths (c).

The output signal $y(i)$ is computed via the temporary output $y_{\mathrm{tmp}}(i) = h(i) * x(i)$ by

$$\begin{aligned} y(i) &= \mathrm{flip}\left(h(i) * y_{\mathrm{tmp}}(-i)\right) && \text{(A.25)}\\ &= h(-i) * y_{\mathrm{tmp}}(i) && \text{(A.26)}\\ &= h(-i) * (h(i) * x(i)). && \text{(A.27)} \end{aligned}$$

It follows in the Fourier domain:

$$\begin{aligned} Y(\mathrm{j}\omega) &= H(-\mathrm{j}\omega) \cdot \left(H(\mathrm{j}\omega) \cdot X(\mathrm{j}\omega)\right) && \text{(A.28)}\\ &= |H(\mathrm{j}\omega)|^2 \cdot X(\mathrm{j}\omega). && \text{(A.29)} \end{aligned}$$

A.3 Correlation Function

The *correlation function* can be used to compute the similarity between two signals at a lag η. Given two signals $x(i)$ and $y(i)$, the so-called *Cross Correlation Function (CCF)* is defined by

$$r_{xy}(\eta) = \sum_{i=-\infty}^{\infty} x(i) \cdot y(i + \eta). \tag{A.30}$$

Assuming the input signals are only blocks of $\mathcal{K} = i_{\mathrm{e}}(n) - i_{\mathrm{s}}(n) + 1$ samples, we can imagine an infinite number of zeros to the right and to the left of the actual samples; Eq. (A.30) can still be applied (although in a real application the multiplications with zero can obviously be omitted) and the CCF of two blocks of samples is thus

$$r_{xy}(\eta, n) = \sum_{i=i_\mathrm{s}(n)}^{i_\mathrm{e}(n)-\eta} x(i) \cdot y(i+\eta) \tag{A.31}$$

and will equal zero for all lags larger than $\mathcal{K} = i_\mathrm{e}(n) - i_\mathrm{s}(n) + 1$:

$$r_{xy}(\eta, n) = 0 \quad \text{if } |\eta| \geq \mathcal{K}. \tag{A.32}$$

Another possible interpretation of the blockwise CCF is the infinite correlation function of signals multiplied with a rectangular window $w_\mathrm{R}(i)$:

$$r_{xy}(\eta, n) = \sum_{i=-\infty}^{\infty} x(i) w_\mathrm{R}(i) \cdot y(i+\eta) w_\mathrm{R}(i+\eta). \tag{A.33}$$

A.3.1 Normalization

The correlation function is usually normalized by a factor λ_c. If only a block of samples of the signals $x(i)$ and $y(i)$ is being considered, then a standard choice for λ_c is

$$\lambda_\mathrm{c} = \frac{1}{\sqrt{\left(\sum\limits_{i=i_\mathrm{s}(n)}^{i_\mathrm{e}(n)} x^2(i)\right) \cdot \left(\sum\limits_{i=i_\mathrm{s}(n)}^{i_\mathrm{e}(n)} y^2(i)\right)}}. \tag{A.34}$$

The number of nonzero multiplications decreases linearly with increasing lag until it is only one for $\eta = K - 1$. Therefore, the resulting correlation function is triangular-shaped with the shape's maximum at lag $\eta = 0$. If required, two workarounds can help avoid this triangular weighting, although they cannot be recommended generally. First, the current lag η can be taken into account for the normalization:

$$\lambda_\mathrm{c}(\eta) = \frac{\mathcal{K}}{(\mathcal{K} - |\eta|) \cdot \sqrt{\left(\sum\limits_{i=i_\mathrm{s}(n)}^{i_\mathrm{e}(n)} x^2(i)\right) \cdot \left(\sum\limits_{i=i_\mathrm{s}(n)}^{i_\mathrm{e}(n)} y^2(i)\right)}}. \tag{A.35}$$

For large lags η the results will, however, become unreliable due to the limited number of samples from which they are calculated.

Second, the triangular shape of the correlation function can be avoided by using signal blocks of different sizes, namely a block length of $3\mathcal{K}$ for the signal x and a block length of $\mathcal{K}$ samples for the signal y. Typically, the number of samples for x is increased by appending $\mathcal{K}$ preceding and succeeding samples, respectively. While this leads to an extended theoretical nonzero range of r_{xy} for $|\eta| \leq 2\mathcal{K}$, the results for $|\eta| > \mathcal{K}$ are usually discarded. The proper normalization procedure is in this case unclear, although three options present themselves, (i) the *normalization per lag* requiring the computation of the denominator of Eq. (A.34) for every individual lag, (ii) an *overall normalization* by computing the denominator from unequal length sequences, and (iii) a *two-stage normalization* by first finding the maximum of the unnormalized cross correlation and then computing the denominator for two short length sequences around the maximum. If only the current block of samples from signal x is available, this block may also be copied and pasted to the start and end of this block, resulting in the same number of $3\mathcal{K}$ samples. If the block of samples from signal x would theoretically be pasted an infinite number of times, then the signal would be periodic with $\mathcal{K}$, which in turn would result in the periodicity of the correlation function: $r_{xy}(\eta) = r_{xy}(\eta + \mathcal{K})$. This is then referred to as the *Circular Correlation Function (CiCF)*; the circularity aspects have also to be taken into account when the CCF is computed in the frequency domain as described in Section A.3.4.

The normalized CCF at lag $\eta = 0$ is also called the *Pearson correlation coefficient* if the means are subtracted from the input signals before calculation.

A.3.2 Autocorrelation Function

The Autocorrelation Function (ACF) is a special case of the correlation function with identical input signals $x(i) = y(i)$ and is therefore referred to as $r_{xx}(\eta)$. It is a measure of self-similarity, and its properties are summarized below.

Figure A.7 shows the ACF of a block of a sinusoidal and a block of white noise.

Autocorrelation Function Properties

- *Autocorrelation Function* (*ACF*) at lag 0:
  $r_{xx}(0, n) = 1$ when normalized according to Eq. (A.34); otherwise, it is the *root mean square (RMS)* (see Section 8.3.1) of the block.
- Maximum:
  $|r_{xx}(\eta, n)| \leq r_{xx}(0, n)$ since the signal can at no lag be more similar to itself than without lag ($\eta = 0$).
- Symmetry:
  $r_{xx}(\eta, n) = r_{xx}(-\eta, n)$. The ACF is symmetric around lag $\eta = 0$. This means that the calculation of negative lags can be omitted for reasons of computational efficiency.
- Periodicity:
  The ACF of a periodic signal will be periodic with the period length of the input signal.

A.3.3 Applications

The CCF can be used to find a specific latency or offset between a signal and the delayed signal. The lag of the maximum of the CCF indicates the delay between the signals. A typical example would be a radar signal with its echo coming back

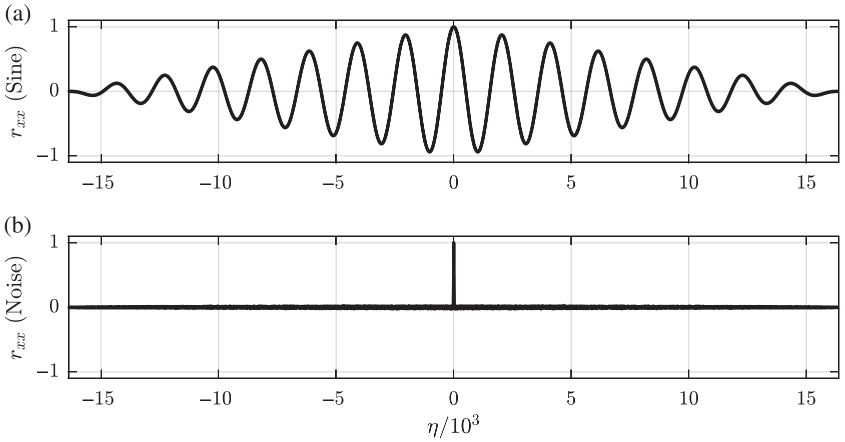


Figure A.7 ACF of a sinusoid (a) and white noise (b).

Repo 1: plotAcfExample.m

after being reflected by an obstacle. Locating the maximum of the CCF between the sent and the reflected signal should indicate the transmission time and thus the distance.

A CCF is sometimes also computed between a real signal and a constructed signal to find out whether the original signal shows similarities to the synthetic signal. An example would be to compute the CCF of the envelope of a drum loop with a series a weighted peaks to find a pattern in the signal.

The ACF is commonly used to find periodicities within the signal. A periodic input signal will lead to a periodic ACF with peaks at the length of the period and its integer multiples. The ACF can be thus used to find a (fundamental) frequency (see Section 7.3.3.2) or to find the time interval between beats in music.

A.3.4 Calculation in the Frequency Domain

Rewriting either the CCF or the ACF as a convolution operation reveals an important property:

$$\begin{aligned} r_{xx}(t) &= \int\limits_{-\infty}^{\infty} x(\tau)\cdot x(t+\tau)\ d\tau \\ &= x(t) * x(-t). \end{aligned} \tag{A.36}$$

Using Eqs. (17.17) and (17.35) it can be deduced that

$$\begin{aligned} \mathfrak{F}\left\{r_{xx}(\tau)\right\} &= R_{xx}(\mathrm{j}\omega) \\ &= \mathfrak{F}\left\{x(\tau) * x(-\tau)\right\} \\ &= X(\mathrm{j}\omega)\cdot X^{*}(\mathrm{j}\omega) \\ &= |X(\mathrm{j}\omega)|^2. \end{aligned} \tag{A.37}$$

This relation is called the *Wiener–Khinchin theorem*.

The computation of the ACF in the time domain can thus be replaced by a simple multiplication in the frequency domain preceded and followed by an FT and an inverse FT, respectively. For long blocks, this will save computing cycles and reduce computational workload when using an Fast Fourier Transform (FFT) algorithm. Note that with blocks of data, the result will be a *circular* ACF if the transformed block of data has not been correctly padded with zeros before transforming it. More specifically, computing the CCF or ACF of two blocks of length $\mathcal{K}$ requires a minimum FFT length of $2\mathcal{K}$.

A.3.4.1 Frequency Domain Compression

In certain applications such as auditory processing it might be of interest to apply a nonlinear compression function to the magnitude spectrum before transforming it back. Tolonen and Karjalainen named such an approach the *generalized ACF* to be computed by (see p. 158, [9])

$$r^{\beta}_{xx}(\eta, n) = \mathfrak{F}^{-1}\left\{|X(\mathrm{j}\omega)|^{\beta}\right\}. \tag{A.38}$$

A value $\beta = 2$ would result in the normal ACF.

References

1 Alan V Oppenheim and Ronald W Schafer. *Discrete-Time Signal Processing*. Prentice Hall, Upper Saddle River, NJ, 2nd edition, 1999. ISBN 0-13-754920-2. 05004.

2 Jens-Rainer Ohm and Hans Dieter Lüke. *Signalübertragung*. Springer-Lehrbuch. Springer, Berlin, Heidelberg, 10th edition, 2007. ISBN 978-3-540-69256-0. URL http://www.springerlink.com/index/10.1007/978-3-540-69258-4.

3 Julius O Smith. Spectral Audio Signal Processing - March 2007 Draft. Technical report, Stanford University, Center for Computer Research in Music and Acoustics (CCRMA), California, CA, 2007.

4 Vladimir Aleksandrovich Kotelnikov. On the Transmission Capacity of 'Ether' and Wire in Electric Communications. *Reprint by the Institute of Radioengineering and Electronics of the Moscow Power Engineering Institute (Technical University) of the original article from 1933*, 2003. doi: 10.1070/PU2006v049n07ABEH006048.

5 Claude E Shannon. Communication In The Presence Of Noise. *Proceedings of the IEEE*, 86(2):447–457, February 1998. ISSN 0018-9219. doi: 10.1109/JPROC.1998.659497. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=659497.

6 John Macnaghten Whittaker. The 'Fourier' Theory of the Cardinal Function. *Proceedings of the Edinburgh Mathematical Society*, 1:169–176, January 1929.

ISSN 0013-0915. doi: 10.1017/S0013091500013511. URL http://www.journals.cambridge.org/abstract_S0013091500013511.

7 Harry Nyquist. Certain Topics in Telegraph Transmission Theory. *Transactions of the American Institute of Electrical Engineers*, 47(2):617–644, April 1928. ISSN 0096-3860. doi: 10.1109/T-AIEE.1928.5055024. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=989875 http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5055024. 01598.

8 Steven W Smith. *Digital Signal Processing (The Scientist's and Engineer's Guide to)*. California Technical Publishing, San Diego, CA, 2nd edition, July 1999. ISBN 978-0-9660176-3-2. URL http://linkinghub.elsevier.com/retrieve/doi/10.1006/dspr.1993.1020.

9 Tero Tolonen and Matti Karjalainen. A Computationally Efficient Multipitch Analysis Model. *Transactions on Speech and Audio Processing*, 8(6):708–716, 2000.

Appendix B

Fourier Transform

The Fourier Transform (FT) is widely used in audio signal analysis and synthesis. Understanding its properties is crucial for the successful design of audio processing systems.

Deriving the FTs fundamental properties is easier for continuous signals; we will thus focus on the continuous domain first and will then discuss the FT of windowed signals, the FT of sampled signals, and finally the Discrete Fourier Transform (DFT).

Periodic signals can be represented as a Fourier series as introduced in Eq. (3.3). The fundamental frequency ω_0 determines the "frequency resolution" of the series. For the analysis of nonperiodic signals, we let the period length grow $T_0 \rightarrow \infty$ (or equivalently $\omega_0 \rightarrow 0$). This has the effect that the previously discrete frequency resolution becomes continuous with $k\omega_0 \rightarrow \omega$. Due to the resulting infinite resolution of the frequency axis, the coefficients will decrease $a_k \rightarrow 0$. The formula for the Fourier series given in Eq. (3.3) thus changes into the FT:

$$X(\mathrm{j}\omega) = \mathfrak{F}\left\{x(t)\right\} = \int\limits_{-\infty}^{\infty} x(t)e^{-\mathrm{j}\omega t}\, dt \tag{B.1}$$

and $X(\mathrm{j}\omega)$ is the so-called *spectrum* of the signal $x(t)$.

The real and imaginary parts represent the even and odd parts of the input signal, extracted with the cosine and shine-shaped basis functions. A common form of showing the results is to represent the spectrum as magnitude $|X(\mathrm{j}\omega)|$ and phase $\Phi_X(\mathrm{j}\omega)$ instead of real and imaginary parts. Frequently, only the *magnitude spectrum* is being used for the analysis or visualization of the spectrum while the phase spectrum is ignored. Another common representation is the *power spectrum* which is the squared magnitude spectrum.

An Introduction to Audio Content Analysis: Music Information Retrieval Tasks & Applications,
Second Edition. Alexander Lerch.
© 2023 The Institute of Electrical and Electronics Engineers, Inc. Published 2023 by John Wiley & Sons, Inc.

B.1 Properties of the Fourier Transformation

This section introduces the most important properties of the Fourier Transform. A summary of these properties formulated for the Short Time Fourier Transform (STFT) for the DFT can be found on Pg. 394.

B.1.1 Inverse Fourier Transform

The FT is invertible. $X(\mathrm{j}\omega)$ can be converted back from the frequency domain into the time domain signal $x(t)$ by applying the *Inverse Fourier Transform (IFT)*:

$$x(t) = \mathfrak{F}^{-1}\left\{X(\mathrm{j}\omega)\right\} = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\mathrm{j}\omega)e^{\mathrm{j}\omega t}\, d\omega. \tag{B.2}$$

That means that time and frequency representation are equivalent, i.e. no information is gained or lost by applying the FT to a signal; it just changes the representation of the signal.

It becomes also obvious from comparing Eqs. (B.1) and (B.2) that forward and inverse transform are very similar operations (see also Sect. B.1.6).

B.1.2 Superposition

If the signal $y(t)$ is the weighted addition of the signals $x_1(t)$ and $x_2(t)$

$$y(t) = c_1 \cdot x_1(t) + c_2 \cdot x_2(t), \tag{B.3}$$

then the same relationship is true for their frequency transformation:

$$\begin{aligned} Y(\mathrm{j}\omega) &= \int_{-\infty}^{\infty} \left(c_1 \cdot x_1(t) + c_2 \cdot x_2(t)\right) \cdot e^{-\mathrm{j}\omega t}\, dt \\ &= c_1 \cdot \int_{-\infty}^{\infty} x_1(t)e^{-\mathrm{j}\omega t}\, dt + c_2 \cdot \int_{-\infty}^{\infty} x_2(t)e^{-\mathrm{j}\omega t}\, dt \\ &= c_1 \cdot X_1(\mathrm{j}\omega) + c_2 \cdot X_2(\mathrm{j}\omega). \end{aligned} \tag{B.4}$$

B.1.3 Convolution and Multiplication

The convolution of signal $x(t)$ with the impulse response $h(t)$

$$\begin{aligned} y(t) &= h(t) * x(t) \\ &= \int_{-\infty}^{\infty} h(\tau) \cdot x(t-\tau)\, d\tau \end{aligned} \tag{B.5}$$

corresponds to a multiplication in the spectral domain

$$Y(\mathrm{j}\omega) = H(\mathrm{j}\omega) \cdot X(\mathrm{j}\omega). \tag{B.6}$$

The derivation involves clever grouping and expansion:

$$\begin{aligned}
Y(\mathrm{j}\omega) &= \int_{-\infty}^{\infty} y(t) e^{-\mathrm{j}\omega t}\, dt \\
&= \int_{-\infty}^{\infty} \left(\int_{-\infty}^{\infty} h(\tau) \cdot x(t-\tau)\, d\tau \right) e^{-\mathrm{j}\omega t}\, dt \\
&= \int_{-\infty}^{\infty} h(\tau) \int_{-\infty}^{\infty} x(t-\tau) e^{-\mathrm{j}\omega t}\, dt\, d\tau \\
&= \int_{-\infty}^{\infty} h(\tau) e^{-\mathrm{j}\omega\tau} \underbrace{\int_{-\infty}^{\infty} x(t-\tau) e^{-\mathrm{j}\omega(t-\tau)}\, d(t-\tau)}_{X(\mathrm{j}\omega)}\, d\tau \\
&= \int_{-\infty}^{\infty} h(\tau) e^{-\mathrm{j}\omega\tau}\, d\tau \cdot X(\mathrm{j}\omega) \\
&= H(\mathrm{j}\omega) \cdot X(\mathrm{j}\omega).
\end{aligned} \tag{B.7}$$

This property allows the efficient computation of the convolution of a signal with an finite impulse response (FIR) filter with a long impulse response in the frequency domain [1].

The same relationship exists for convolution in the frequency domain. The convolution operation

$$Y(\mathrm{j}\omega) = H(\mathrm{j}\omega) * X(\mathrm{j}\omega) \tag{B.8}$$

could be replaced by a multiplication in the time domain

$$y(t) = h(t) \cdot x(t). \tag{B.9}$$

B.1.4 Parseval's Theorem

The energy of the signal can be calculated in both the time and the spectral domain:

$$\int_{-\infty}^{\infty} x^2(t)\, dt = \frac{1}{2\pi} \int_{-\infty}^{\infty} |X(\mathrm{j}\omega)|^2\, d\omega. \tag{B.10}$$

This can be shown by using the equivalence between multiplication in the frequency domain and convolution in the time domain. Writing

$$\int_{-\infty}^{\infty} h(\tau) \cdot x(t-\tau)\, d\tau = \frac{1}{2\pi} \int_{-\infty}^{\infty} H(\mathrm{j}\omega) \cdot X(\mathrm{j}\omega) e^{\mathrm{j}\omega t} d\omega \tag{B.11}$$

and replacing $H(\mathrm{j}\omega)$ with the conjugate-complex spectrum $X^*(\mathrm{j}\omega)$ and $h(\tau)$ with $x(-\tau)$,[1] respectively, the result at $t = 0$ is

$$\int_{-\infty}^{\infty} x(-\tau) \cdot x(-\tau)\, d\tau = \frac{1}{2\pi} \int_{-\infty}^{\infty} X^*(\mathrm{j}\omega) \cdot X(\mathrm{j}\omega)\, d\omega,$$

$$\int_{-\infty}^{\infty} x^2(t)\, dt = \frac{1}{2\pi} \int_{-\infty}^{\infty} |X(\mathrm{j}\omega)|^2\, d\omega. \tag{B.12}$$

B.1.5 Time and Frequency Shift

The transform of a signal shifted by a constant in time $y(t) = x(t - t_0)$ is

$$Y(\mathrm{j}\omega) = X(\mathrm{j}\omega)e^{-\mathrm{j}\omega t_0}. \tag{B.13}$$

This means that the magnitude spectrum will be identical but the phase spectrum will have a linear offset $\Phi_Y(\omega) = \Phi_X(\omega) - \omega t_0$:

$$\begin{aligned}\int_{-\infty}^{\infty} x(t - t_0)e^{-\mathrm{j}\omega t}\, dt &= \int_{-\infty}^{\infty} x(\tau)e^{-\mathrm{j}\omega(\tau + t_0)}\, d\tau \\ &= e^{-\mathrm{j}\omega t_0} \int_{-\infty}^{\infty} x(\tau)e^{-\mathrm{j}\omega\tau}\, d\tau \\ &= e^{-\mathrm{j}\omega t_0} \cdot X(\mathrm{j}\omega).\end{aligned} \tag{B.14}$$

Equivalently, the shifted spectrum[2] $Y(\mathrm{j}\omega) = X(\mathrm{j}(\omega - \omega_0))$ corresponds to the time domain signal $y(t) = x(t) \cdot e^{\mathrm{j}\omega_0 t}$ which is the original signal modulated by a sinusoidal signal:

$$\begin{aligned}\frac{1}{2\pi} \int_{-\infty}^{\infty} X(\mathrm{j}\omega - \omega_0)e^{\mathrm{j}\omega t}\, d\omega &= \frac{1}{2\pi} \int_{-\infty}^{\infty} X(\mathrm{j}\phi)e^{\mathrm{j}(\phi + \omega_0)t}\, d\phi \\ &= e^{\mathrm{j}\omega_0 t} \cdot x(t).\end{aligned} \tag{B.15}$$

B.1.6 Symmetry

If the time domain signal $x(t)$ is real-valued, then its frequency transform will be symmetric with $X(\mathrm{j}\omega) = X^*(-\mathrm{j}\omega)$. The magnitude is symmetric around the y-axis:

$$|X(\mathrm{j}\omega)| = |X(-\mathrm{j}\omega)| \tag{B.16}$$

---

[1] Only real-valued time domain functions $x(t)$ are considered here.

[2] In real-valued time signals, this shift has to be applied symmetrically to the negative frequencies.

while the phase is symmetric around the origin:

$$\Phi_X(\omega) = -\Phi_X(-\omega). \tag{B.17}$$

Vice versa, if the frequency transform is real-valued, then the time domain signal will be symmetric with $x(t) = x(-t)$ and if $X(\mathrm{j}\omega)$ is imaginary it means that $x(t) = -x(-t)$. This can be shown by representing the time signal $x(t)$ as a sum of an even component x_e and an odd component $x_\mathrm{o}(t)$:

$$x(t) = \underbrace{\frac{1}{2}(x(t) + x(-t))}_{x_\mathrm{e}(t)} + \underbrace{\frac{1}{2}(x(t) - x(-t))}_{x_\mathrm{o}(t)}. \tag{B.18}$$

The FT of the even and odd signal components is then

$$X_\mathrm{e}(\mathrm{j}\omega) = \int_{-\infty}^{\infty} x_\mathrm{e}(t)\cos(\omega t)\, dt - \mathrm{j}\underbrace{\int_{-\infty}^{\infty} x_\mathrm{e}(t)\sin(\omega t)\, dt}_{=0}, \tag{B.19}$$

$$X_\mathrm{o}(\mathrm{j}\omega) = \underbrace{\int_{-\infty}^{\infty} x_\mathrm{o}(t)\cos(\omega t)\, dt}_{=0} - \mathrm{j}\int_{-\infty}^{\infty} x_\mathrm{o}(t)\sin(\omega t)\, dt. \tag{B.20}$$

The transform of the even signal is thus purely real $X_\mathrm{e}(\mathrm{j}\omega) = \mathfrak{Re}[X(\mathrm{j}\omega)]$, and the transform of the odd part is purely imaginary $X_\mathrm{o}(\mathrm{j}\omega) = \mathfrak{Im}[X(\mathrm{j}\omega)]$. Furthermore, due to the property $\cos(\omega t) = \cos(-\omega t)$, it becomes clear that the real part is again an even function symmetric around $\omega = 0$. The imaginary part is odd due to $\sin(\omega t) = -\sin(-\omega t)$. It follows that the magnitude spectrum is an even function and the phase spectrum is an odd function.

We have seen that the FT is very similar to the IFT; thus, if $X(\mathrm{j}\omega)$ is the FT of the signal $x(t)$, then it would also be true that $2\pi \cdot x(-\mathrm{j}\omega)$ is the transform of $X(t)$. This can be shown by substituting t with $-\omega$ in the IFT:

$$\begin{aligned} x(t) &= \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\mathrm{j}\omega)e^{\mathrm{j}\omega t}\, d\omega, \\ x(-t) &= \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\mathrm{j}\omega)e^{-\mathrm{j}\omega t}\, d\omega, \\ x(-\mathrm{j}\omega) &= \frac{1}{2\pi}\int_{-\infty}^{\infty} X(t)e^{-\mathrm{j}\omega t}\, dt. \end{aligned} \tag{B.21}$$

B.1.7 Time and Frequency Scaling

The FT of a signal modified in the time domain by scaling the time axis $y(t) = x(c \cdot t)$ will be scaled inversely:

$$Y(\mathrm{j}\omega) = \frac{1}{|c|} X\left(\mathrm{j}\frac{\omega}{c}\right). \tag{B.22}$$

The derivation (for positive c) is

$$\begin{aligned} Y(\mathrm{j}\omega) &= \int_{-\infty}^{\infty} x(c \cdot t) e^{-\mathrm{j}\omega t}\, dt \\ &= \int_{-\infty}^{\infty} x(\tau) e^{-\mathrm{j}\omega \frac{\tau}{c}}\, d\frac{\tau}{c} \\ &= \frac{1}{c} \int_{-\infty}^{\infty} x(\tau) e^{-\mathrm{j}\frac{\omega}{c}\tau}\, d\tau \\ &= \frac{1}{c} X\left(\mathrm{j}\frac{\omega}{c}\right). \end{aligned} \tag{B.23}$$

For negative c, the result is $Y(\mathrm{j}\omega) = -\frac{1}{c} X(\mathrm{j}\frac{\omega}{c})$. The spectrum of a stretched signal ($c > 1$) will thus be compressed and vice versa.

From the above equation, it directly follows for $c = -1$ that

$$\mathfrak{F}\left\{x(-t)\right\} = X(-\mathrm{j}\omega) \tag{B.24}$$

and for a real-valued signal $x(t)$

$$\mathfrak{F}\left\{x(-t)\right\} = X^*(\mathrm{j}\omega). \tag{B.25}$$

B.1.8 Derivatives

The transform of the nth derivative of the signal has the following property (without derivation):

$$\mathfrak{F}\left\{\frac{d^n x(t)}{dt^n}\right\} = (\mathrm{j}\omega)^n X(\mathrm{j}\omega). \tag{B.26}$$

B.2 Spectrum of Example Time Domain Signals

B.2.1 Delta Function

The *delta function* $\delta(t)$, sometimes also named *dirac impulse* or *delta impulse*, equals zero for all points in time except $t = 0$. It represents an ideal impulse and is defined by

$$\int_{-\infty}^{\infty} \delta(t)\, dt = 1, \tag{B.27}$$

$$\delta(t) = 0 \text{ for all } t \neq 0. \tag{B.28}$$

This also means that the integration of the multiplication of signal $x(t)$ with this delta function results in

$$\int_{-\infty}^{\infty} x(t) \cdot \delta(t)\, dt = x(0). \tag{B.29}$$

Thus, the result of the FT is

$$\Delta(\mathrm{j}\omega) = \int_{-\infty}^{\infty} \delta(t) e^{-\mathrm{j}\omega t}\, dt = e^{\mathrm{j}\omega \cdot 0} = 1. \tag{B.30}$$

The spectrum is therefore a real-valued constant; it follows that the delta function incorporates all frequencies with the same strength.

B.2.2 Constant

The symmetry of FT and IFT shown in Eq. (B.21), in combination with Eq. (B.30), also tells us that the spectrum of a constant valued time domain signal $x(t) = {}^{1}\!/_{2\pi}$ will be $X(\mathrm{j}\omega) = \delta(\omega)$.

B.2.3 Cosine

A sinusoidal time domain signal can be interpreted as a modulated constant value. Applying the frequency shift property from Eq. (B.15) thus shows that the spectrum of a cosine is the spectrum of a constant value shifted by the cosine's frequency ω_0, the delta function $\delta(\omega - \omega_0)$.

B.2.4 Rectangular Window

The rectangular window is defined by

$$w_\mathrm{R}(t) = \begin{cases} 1, & -\frac{1}{2} \leq t \leq \frac{1}{2} \\ 0, & \text{otherwise} \end{cases}. \tag{B.31}$$

The spectrum of this window function is

$$\begin{aligned} W_\mathrm{R}(\mathrm{j}\omega) &= \int_{-\infty}^{\infty} w_\mathrm{R}(t) e^{-\mathrm{j}\omega t}\, dt \\ &= \int_{-1/2}^{1/2} e^{-\mathrm{j}\omega t}\, dt \\ &= \frac{1}{-\mathrm{j}\omega} \underbrace{\left(e^{-\mathrm{j}\omega/2} - e^{\mathrm{j}\omega/2}\right)}_{=-2j\sin(\omega/2)} \\ &= \frac{\sin(\omega/2)}{\omega/2} = \operatorname{sinc}\left(\frac{\omega}{2}\right). \end{aligned} \tag{B.32}$$

B.2.5 Delta Pulse

The *delta pulse* is a series of individual delta impulses, i.e. a superposition of delta functions. It is defined by

$$\delta_{\mathrm{T}}(t) = \sum_{i=-\infty}^{\infty} \delta(t - iT_0). \tag{B.33}$$

Each delta impulse has a distance T_0 from its neighbor. Using FT of a delta function given by Eq. (B.27) and the superposition property from Eq. (B.3) in combination with the time shift property from Eq. B.13, the FT of $\delta_{\mathrm{T}}(t)$ is

$$\Delta_{\mathrm{T}}(\mathrm{j}\omega) = \sum_{i=-\infty}^{\infty} e^{-\mathrm{j}\omega i T_0}. \tag{B.34}$$

With help from the geometric series it can be shown [2] that

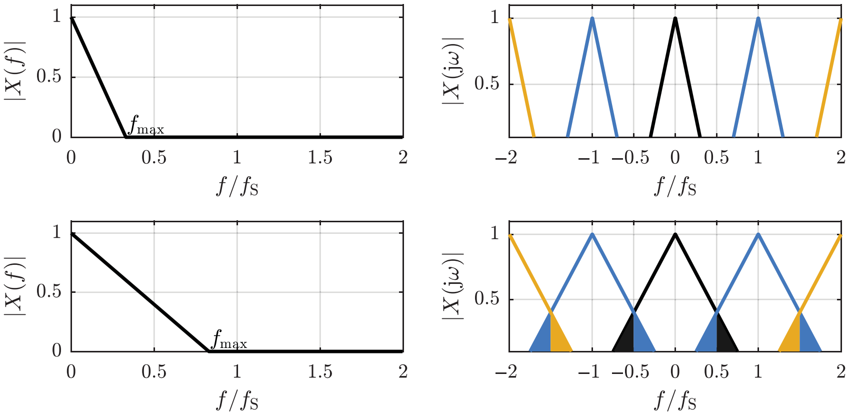
$$\begin{aligned}\Delta_{\mathrm{T}}(\mathrm{j}\omega) &= \frac{2\pi}{T} \sum_{i=-\infty}^{\infty} \delta\left(\omega - \frac{2\pi i}{T}\mathrm{j}\omega T\right)\\ &= \omega_T \delta_{\omega_{\mathrm{T}}}(\omega).\end{aligned} \tag{B.35}$$

B.3 Transformation of Sampled Time Signals

A sampled time signal can be represented by the multiplication of a continuous time signal $x(t)$ by a delta pulse $\delta_{\mathrm{T}}(t)$. Equation (B.8) states that a multiplication of two time signals corresponds to the convolution of their frequency transforms. This means that

$$\begin{aligned}\mathfrak{F}\{x(i)\} &= \mathfrak{F}\left\{x(t) \cdot \delta_{\mathrm{T}}(t)\right\}\\ &= \mathfrak{F}\{x(t)\} * \mathfrak{F}\left\{\delta_{\mathrm{T}}(t)\right\}\\ &= X(\mathrm{j}\omega) * \Delta_{\mathrm{T}}(\mathrm{j}\omega).\end{aligned} \tag{B.36}$$

Note that although the time domain signal is discrete, the resulting spectrum is still continuous. As can be seen from Eq. (B.36), the spectrum is repeated periodically with ω_{T}, the sample rate. This allows a very intuitive explanation of the sampling theorem stated in Eq. (16.2) since the periodically repeated spectra would overlap if signal $x(t)$ contains higher frequencies than $\omega_{\mathrm{T}}/2$ (see Figure B.1). In that case, reconstruction of the original signal $x(t)$ is impossible, while otherwise perfect reconstruction is possible by applying an ideal low-pass filter with a cut-off frequency of $\omega_{\mathrm{T}}/2$ to the sampled signal $x(i)$. The effect of overlapping spectra is called *aliasing* and is visualized in Figure B.1.



Repo 1: plotSpectralOverlap.m

Figure B.1 Schematic visualization of the spectrum of a continuous time domain signal (left) and the sampled signal in accordance (top) and violation (bottom) of the sampling theorem; areas with aliasing are filled.

B.4 Short Time Fourier Transform of Continuous Signals

Up to this point, we have dealt mostly with signals unlimited in time. In the real world, signals will usually have a defined start and stop time. We might also be interested in transforming only segments of such signals. This can be interpreted as multiplying an infinite time signal with a window function that equals zero outside the time boundaries of interest. In signal analysis, typical segment lengths range – dependent on the task at hand – between 10 and 300 ms. Smith points out three reasons for choosing segments of this length [3]:

- “Perhaps most fundamentally, the ear similarly Fourier analyzes only a short segment of audio signals at a time (on the order of 10–20 ms worth). Therefore, to match our spectrum analysis to human hearing, we desire to limit the time window of the analysis.”
- “Audio signals typically have spectra which change over time. It is therefore usually most meaningful to restrict analysis to a time window over which the spectrum stays rather constant.”
- “It can be extremely time consuming to compute the Fourier transform of an audio signal of typical length, and it will rarely fit in computer memory all at once.”

Fourier Transform Properties

- Invertibility

$$x\big(i_{\mathrm{s}}(n)\dots i_{\mathrm{e}}(n)\big) = \mathfrak{F}^{-1}\left\{X(k,n)\right\} = \frac{1}{\mathcal{K}}\sum_{0}^{\mathcal{K}-1} X(k,n)\cdot \exp\left(\mathrm{j}ki\frac{2\pi}{\mathcal{K}}\right). \tag{B.37}$$

- Linearity and Superposition

$$\begin{aligned}&\mathfrak{F}\left\{c_1\cdot x\left(i_{\mathrm{s}}(n)\dots i_{\mathrm{e}}(n)\right) + c_2\cdot y\left(i_{\mathrm{s}}(n)\dots i_{\mathrm{e}}(n)\right)\right\}\\ &\qquad = c_1\cdot X(k,n) + c_2\cdot Y(k,n).\end{aligned} \tag{B.38}$$

- Periodicity

$$X(k+\mathcal{K},n) = X(k,n). \tag{B.39}$$

- Symmetry [for real $x(i)$]

$$X(\mathcal{K}-k,n) = X^{*}(k,n). \tag{B.40}$$

- Circularity

$$\mathfrak{F}^{-1}\left\{H(k,n)\cdot X(k,n)\right\} = h'\left(i_{\mathrm{s}}(n)\dots i_{\mathrm{e}}(n)\right) * x\left(i_{\mathrm{s}}(n)\dots i_{\mathrm{e}}(n)\right) \tag{B.41}$$

  with h' being a *periodically extended* sequence of the signal block with the sample boundaries $i_{\mathrm{s}}(n)$ and $i_{\mathrm{e}}(n)$.

- Time and Frequency Shifting

$$\mathfrak{F}\left\{x(i-i_0)\right\} = X(k,n)\cdot \exp\left(-\mathrm{j}\frac{2\pi k}{\mathcal{K}} i_0\right) \tag{B.42}$$

$$\mathfrak{F}^{-1}\left\{X(k-k_0,n)\right\} = x\left(i_{\mathrm{s}}(n)\dots i_{\mathrm{e}}(n)\right)\cdot \exp\left(\mathrm{j}\frac{2\pi i}{\mathcal{K}} k_0\right). \tag{B.43}$$

- Time and Frequency Scaling

$$\mathfrak{F}\left\{x(c\cdot i)\right\} = \frac{1}{|c|} X\left(\frac{k}{c}, n\right). \tag{B.44}$$

- Parseval's Theorem

$$\sum_{i=i_{\mathrm{s}}(n)}^{i_{\mathrm{e}}(n)} |x(i)|^2 = \frac{1}{\mathcal{K}}\sum_{k=0}^{\mathcal{K}-1} |X(k,n)|^2. \tag{B.45}$$

- Duality

$$\mathfrak{F}\left\{X(i)\right\} = \frac{1}{\mathcal{K}} x(k,n). \tag{B.46}$$

B.4.1 Window Functions

Since every multiplication in the time domain corresponds to a convolution of the corresponding spectra, the spectrum of the signal is convolved with the spectrum of the window function. The spectrum of the window thus has influence on the resulting spectrum. The most simple window in the time domain is a rectangular window introduced in Sect. B.2.4. The typical spectral shape of a window consists of a main lobe and many side lobes with more or less decreasing amplitude.

When the signal $x(t)$ of interest is a sinusoid, then the resulting FT of the windowed signal will therefore be a superposition of two window functions with their main lobes located at the signal's frequency $\omega_0, -\omega_0, \ldots$, so the delta functions are effectively "smeared" by windowing artifacts. This undesired side effect is referred to as *spectral leakage*. It is usually characterized by

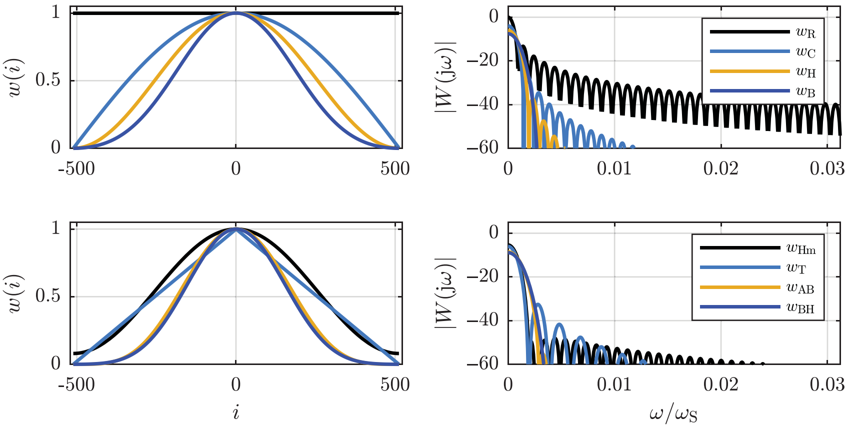
- the width of the main lobe,
- the height of the first (closest) side lobe peak, and
- the roll-off or attenuation of the side lobe peaks.

In order to optimize these properties toward individual use cases, different window functions have been suggested in the past. Figure B.2 shows common window functions in time domain (left) and frequency domain (right).

B.4.1.1 Rectangular Window

The FT of the *rectangular window* has already been derived in Sect. B.2.4 to be a sinc function:

$$W_{\mathrm{R}}(\mathrm{j}\omega) = \operatorname{sinc}\left(\frac{\omega}{2}\right) \tag{B.47}$$



Repo 1: plotSpectralWindows.m

Figure B.2 Windows in time domain (left) and frequency domain (right).

B.4.1.2 Bartlett Window

The *Bartlett window* has a triangular shape. It is defined by

$$\begin{aligned} w_{\mathrm{T}}(t) &= \begin{cases} t+1, & -1/2 \leq t \leq 0 \\ 1-t, & 0 \leq t \leq 1/2 \\ 0, & \text{otherwise} \end{cases} \\ &= w_{\mathrm{R}}(2t) * w_{\mathrm{R}}(2t). \end{aligned} \tag{B.48}$$

Using Eq. (B.7) it can be deduced that

$$\begin{aligned} W_{\mathrm{T}}(\mathrm{j}\omega) &= \mathfrak{F}\left\{w_{\mathrm{R}}(2t)\right\} \cdot \mathfrak{F}\left\{w_{\mathrm{R}}(2t)\right\} \\ &= \frac{1}{2} \cdot \operatorname{sinc}^2\left(\frac{\omega}{4}\right) \end{aligned} \tag{B.49}$$

B.4.1.3 Generalized Superposed Cosines

It is possible to generalize many window functions with

$$w_{\sup}(t) = w_{\mathrm{R}}(t) \sum_{j=0}^{\mathcal{O}-1} b_j \cos\left(\frac{\pi}{2} j t\right). \tag{B.50}$$

Different values for $\mathcal{O}$ result in different window families:

- $\mathcal{O} = 1$: *rectangular window* $w_{\mathrm{R}}(t)$
- $\mathcal{O} = 2$: Hamming family of windows:
  - *cosine window* $w_{\mathrm{C}}(t)$:
    $b_0 = 0,\ b_1 = 1$
  - *von-Hann window* $w_{\mathrm{H}}(t)$:
    $b_0 = 1/2,\ b_1 = 1/2$
  - *Hamming window* $w_{\mathrm{Hm}}(t)$:
    $b_0 = 25/46,\ b_1 = 42/46$
- $\mathcal{O} = 3$
  - classic *Blackman window* $w_{\mathrm{B}}(t)$:
    $b_0 = 7938/18608,\ b_1 = 9240/18608,\ b_2 = 1430/18608$
- $\mathcal{O} = 4$
  - *Blackman–Harris window* $w_{\mathrm{BH}}(t)$:
    $b_0 = 0.35875,\ b_1 = 0.48829,\ b_2 = 0.14128, b_3 = 0.01168$

B.4.1.4 Generalized Power of Cosine

A different generalization of window functions is (see [3])

$$w_{\mathrm{pow}}(t) = w_{\mathrm{R}}(t) \cos^{\beta}\left(\frac{\pi}{2} t\right). \tag{B.51}$$

Again, different windows can be derived for different β:

- *rectangular window* $w_{\mathrm{R}}(t)$:
  $\beta = 0$
- *cosine window* $w_{\mathrm{C}}(t)$:
  $\beta = 1$

- *von-Hann window* $w_{\mathrm{H}}(t)$:
  $\beta = 2$
- *alternative Blackman window* $w_{\mathrm{AB}}(t)$:
  $\beta = 4$

B.5 Discrete Fourier Transform

In computer applications, a discrete representation of the signal's spectrum is required; it can only be defined at discrete frequency bins. The frequency bins are evenly distributed over the interesting range of frequencies with the distance

$$\Delta\Omega = \frac{2\pi}{\mathcal{K}}. \tag{B.52}$$

The DFT of the nth block of the signal $x(i)$ will be referred to as STFT and is defined by

$$X(k) = \sum_{i=0}^{\mathcal{K}-1} x(i) \exp\left(-\mathrm{j}ki\frac{2\pi}{\mathcal{K}}\right) \tag{B.53}$$

with $k = 0, 1, \ldots, \mathcal{K} - 1$.

Thus, the DFT of a block of samples of length $\mathcal{K}$ also consists of exactly $\mathcal{K}$ complex values; however, since the signal $x(i)$ is real, the result will be symmetric with

$$X(\mathcal{K} - k) = X^*(k) \tag{B.54}$$

and only $\mathcal{K}/2 + 1$ complex results need to be stored.

The spectrum $X(k, n)$ can be interpreted as the (continuous) FT of the block n of signal $x(i)$ sampled at equidistant bins at the positions $k \cdot \Delta\Omega$. It has to be periodic

$$X(k) = X(k + \mathcal{K}) \tag{B.55}$$

because the time domain signal is discrete.

The spectrum can only be discrete if the time domain signal is periodic (compare the Fourier series). Therefore, the DFT can be interpreted as the FT applied to the current block of samples periodically continued.

The *Inverse Discrete Fourier Transform (IDFT)* allows reconstruction of the time samples that had been transformed:

$$x(i) = \sum_{k=0}^{\mathcal{K}-1} X(k) e^{\mathrm{j}ki\Delta\Omega}. \tag{B.56}$$

The properties of the DFT correspond to the properties introduced for the continuous FT, but a few details have to be kept in mind: the multiplication of two DFTs corresponds to a circular convolution (similar to the CiCF) in the time domain. The same is true for time and frequency shift operations. Proper zero-padding the signal before the frequency transform can help in avoiding the circularity effects.

B.5.1 Window Functions

The discrete window functions are sampled (and potentially shifted) versions of the continuous window functions given above. The rectangular window is

$$w_{\mathrm{R}}(i) = \begin{cases} 1, & -\frac{\mathcal{K}-1}{2} \leq i \leq \frac{\mathcal{K}-1}{2} \\ 0, & \text{otherwise} \end{cases}, \tag{B.57}$$

and the superposed cosine window is

$$w_{\mathrm{sup}}(i) = w_{\mathrm{R}}(i) \sum_{j=0}^{\mathcal{O}-1} a_j \cos\left(\frac{j \cdot \pi}{\mathcal{K}} i\right). \tag{B.58}$$

The DFT of a rectangular window is

$$W_{\mathrm{R}}(k, n) = \exp\left(-\mathrm{j}\frac{\mathcal{K}-1}{2}\frac{2\pi k}{\mathcal{K}}\right) \cdot \frac{\sin\left(\frac{\mathcal{K}\frac{2\pi k}{\mathcal{K}}}{2}\right)}{\sin\left(\frac{\frac{2\pi k}{\mathcal{K}}}{2}\right)}. \tag{B.59}$$

Note that the phase shift term originates in moving the first window sample to sample 0. When transforming a windowed sine, the result will be shifted in the frequency domain

$$X(k, n) = \exp\left(-\mathrm{j}\frac{\mathcal{K}-1}{2}\frac{2\pi k}{\mathcal{K}} - \Omega_0\right) \cdot \frac{\sin\left(\frac{\mathcal{K}\frac{2\pi k}{\mathcal{K}}}{2} - \Omega_0\right)}{\sin\left(\frac{\frac{2\pi k}{\mathcal{K}}}{2} - \Omega_0\right)}. \tag{B.60}$$

If the sinusoidal frequency exactly fits the frequency of a bin with index k, then all bins will be zero for $k \neq k_0$. In this case, all the zero crossings of the window function fall at the spectral bin positions. However, if k_0 is between two frequency bins, then two artifacts appear: the main peak has lower level, the so-called *process loss*, and the frequency bins are now directly located at the main peaks of the side lobes. Two special cases are the *best case* and the *worst-case* scenarios with k_0 being directly on a bin or exactly between two bins. In the time domain, the best case means that one or more periods of the sinusoidal fit exactly into the window with length $\mathcal{K}$.

B.5.1.1 Discrete Window Properties

The following properties can be used to characterize the frequency domain representation of a window function:

- width of main lobe ΔB (3 dB bandwidth in bins),
- peak level of highest side lobe A_{SL} (dB),
- side lobe fall-off S_{SL} (dB/Oct),
- worst-case process loss A_{WC} (dB).

Table B.1 Frequency domain properties of the most common windows.

Window	ΔB [Bins]	A_{SL} [dB]	S_{SL} [dB/Oct]	A_{WC} [dB]
w_{R}	0.89	−13	−6	3.92
w_{T}	1.28	−27	−12	3.07
w_{C}	1.20	−23	−12	3.01
w_{H}	1.44	−32	−18	3.18
w_{Hm}	1.30	−43	−6	3.10
w_{B}	1.68	−58	−18	3.47
w_{AB}	1.66	−39	−24	3.47
w_{BH}	1.66	−67	−6	3.45

Source: Data from Harris [4].

A smaller main lobe width yields better frequency resolution and both a smaller side lobe peak level and higher side lobe fall-off results in less cross-talk between sinusoids of different frequencies. The smaller the worst-case process loss, the higher the resulting amplitude accuracy.

Table B.1 summarizes these properties for common windows. For detailed introductions to spectral leakage see Harris [4] and Smith [3].

B.5.2 Fast Fourier Transform

An efficient way to compute the DFT is the *Fast Fourier Transform (FFT)*. The FFT is equivalent to the “normal” DFT; it just computes the result more efficiently. More specifically, the number of operations is approximately $O(\mathcal{K}^2)$ for the normal DFT compared to $O(\mathcal{K}\log\mathcal{K})$ for the FFT (see [5, 6]). There are different algorithms to compute the FFT; most FFT implementations require an input block length which equals a power of 2.

B.6 Frequency Reassignment: Instantaneous Frequency

Frequency reassignment is a method to virtually improve the spectral resolution by mapping the frequency data to coordinates closer to its “true” coordinates. Frequency reassignment is not a fundamental frequency detection technique but can help to increase the frequency accuracy of spectral frequency analysis by utilizing the phase spectrum.

The reassignment process may cover both frequency reassignment as well as time reassignment. For an overview of (time-) frequency reassignment, see Fulop and Fitz [7] and Lagrange and Marchand [8]. The focus will be exclusively on frequency reassignment in the following.

The basic idea of frequency reassignment is that it is possible to estimate the frequency of a signal via its phase. The frequency is the derivative of the phase $\Phi(k,t)$:

$$\omega(k,t) = \frac{d}{dt}\Phi(k,t). \tag{B.61}$$

This is visualized in Figure B.3 using the phasor representation of a sinusoidal, where the length of the vector represents the amplitude, the angle the phase, and the time for one complete rotation the period length. This means that knowing the phase difference $\Delta\Phi$ over a measurement time, we can derive what time it would take the phasor to make a full rotation, which is the period length.

Two common approaches are used to compute the so-called *instantaneous frequency* from the discrete STFT. The first one is to literally compute the phase difference of consecutive STFT spectra, and the second one is to compute two STFTs with different windows, namely with the second window being the derivative of the first window.

The derivation operation from Eq. (B.61) can be easily approximated by computing the difference of the phases of consecutive STFT blocks:

$$\omega_\mathrm{I}(k,n) = \frac{\Delta\Phi_\mathrm{u}(k,n)}{\mathcal{H}}\cdot f_\mathrm{S}. \tag{B.62}$$

The actual computation, however, is not as trivial as it seems at first glance as $\Delta\Phi_\mathrm{u}(k,n)$ represents the *unwrapped* phase difference while the computed phase spectrum gives the *wrapped* phase in the range of $]-\pi,\pi]$. In order to estimate

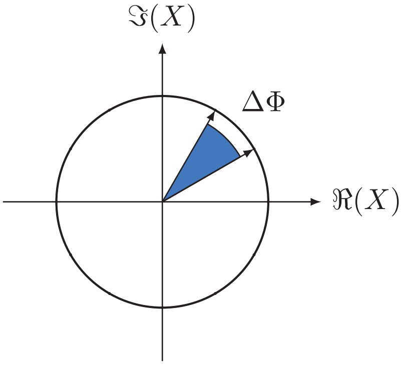


Figure B.3 Phasor representation and visualization of the phase difference when rotating.

the unwrapped phase difference, we first compute the estimate of the unwrapped phase of the current analysis block by

$$\hat{\Phi}(k,n) = \Phi(k,n-1) + \frac{2\pi k}{\mathcal{K}}\mathcal{H}. \tag{B.63}$$

Since the phase $\Phi(k,n)$ is in the range of $]-\pi,\pi]$, the unwrapped phase can then be formulated as follows:

$$\Phi_\mathrm{u}(k,n) = \hat{\Phi}(k,n) + \mathrm{princarg}\left[\Phi(k,n) - \hat{\Phi}(k,n)\right] \tag{B.64}$$

with the princarg function being the principal argument of the phase with a resulting range of $]-\pi,\pi]$. The phase difference is then

$$\begin{aligned}\Delta\Phi_\mathrm{u}(k,n) &= \Phi_\mathrm{u}(k,n) - \Phi(k,n-1)\\ &= \hat{\Phi}(k,n) + \mathrm{princarg}\left[\Phi(k,n) - \hat{\Phi}(k,n)\right] - \Phi(k,n-1).\end{aligned}$$

The final result can be retrieved by substituting $\hat{\Phi}(k,n)$ with Eq. (B.63):

$$\Delta\Phi_\mathrm{u}(k,n) = \frac{2\pi k}{\mathcal{K}}\mathcal{H} + \mathrm{princarg}\left[\Phi(k,n) - \Phi(k,n-1) - \frac{2\pi k}{\mathcal{K}}\mathcal{H}\right]. \tag{B.65}$$

To improve the reliability of the phase unwrapping process the hop size $\mathcal{H}$ should be as small as possible.

An alternative to computing the phase difference directly is to use two different windows for computing two STFTs of one analysis block. Then, the instantaneous frequency can be computed by

$$\omega_\mathrm{I}(k,n) = \omega(k,n) + \Im\left\{\frac{X_D(k,n)\cdot X^*(k,n)}{|X(k,n)|^2}\right\} \tag{B.66}$$

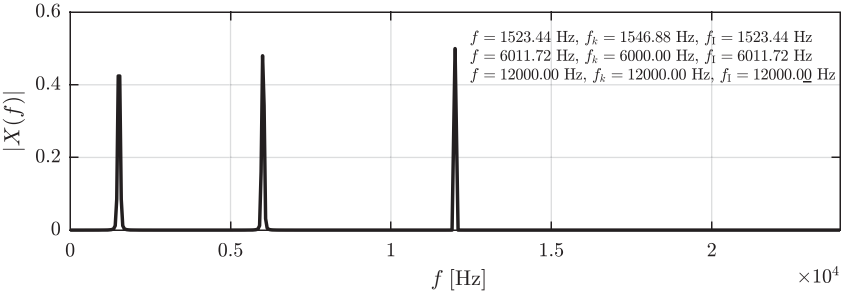


Figure B.4 Magnitude spectrum of a signal composed of three sinusoidals with the input frequency, the frequency of the maximum bin, and the computed instantaneous frequency.

Repo 1: plotInstantaneousFreq.m

with $X_D(k, n)$ being the STFT computed using the derivative of the window used for the calculation of $X(k, n)$ [9]. This approach is independent of the hop size but requires the computation of two DFTs per block.

Figure B.4 shows the effect of computing the instantaneous frequency on a mixture of three sinusoidals, one which falls exactly between frequency bins ($f = 1523.44$ Hz), one which falls directly on a frequency bin ($f = 12$ kHz), and one that is a quarter bin away from the closest frequency bin ($f = 6011.72$ Hz), naming the actual frequencies, the frequency of the closest bin, and the computed instantaneous frequencies.

References

1 William G Gardner. Efficient Convolution without Input-Output Delay. *Journal of the Audio Engineering Society (JAES)*, 43(3):127–136, 1995.

2 Jens-Rainer Ohm and Hans Dieter Lüke. *Signalübertragung*. Springer-Lehrbuch. Springer, Berlin, Heidelberg, 10th edition, 2007. ISBN 978-3-540-69256-0. URL http://www.springerlink.com/index/10.1007/978-3-540-69258-4.

3 Julius O Smith. Spectral Audio Signal Processing - March 2007 Draft. Technical report, Stanford University, Center for Computer Research in Music and Acoustics (CCRMA), California, CA, 2007.

4 Fredric J Harris. On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform. *Proceedings of the IEEE*, 66(1):51–83, 1978.

5 James W Cooley and John W Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*, 19(90):297, April 1965. ISSN 00255718. doi: 10.2307/2003354. URL http://www.jstor.org/stable/2003354?origin=crossref.

6 William T Cochran, James W Cooley, David L Favin, Howard D Helms, Reginald A Kaenel, William W Lang, George C Maling, David E Nelson, Charles M Rader, and Peter D Welch. What is the Fast Fourier Transform? *Transactions on Audio and Electroacoustics*, 15(2):45–55, June 1967. ISSN 0018-9278. doi: 10.1109/TAU.1967.1161899. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1161899.

7 Sean A Fulop and Kelly Fitz. Algorithms for Computing the Time-corrected Instantaneous Frequency (Reassigned) Spectrogram, with Applications. *Journal of the Acoustical Society of America (JASA)*, 119(1):360–371, 2006. ISSN 00014966. doi: 10.1121/1.2133000. URL http://link.aip.org/link/JASMAN/v119/i1/p360/s1&Agg=doi.

8 Mathieu Lagrange and Sylvain Marchand. Estimating the Instantaneous Frequency of Sinusoidal Components Using Phase-Based Methods. *Journal of the Audio Engineering Society (JAES)*, 55(5):385–399, 2007.

9 Francois Auger and Patrick Flandrin. Improving the Readability of Time-frequency and Time-scale Representations by the Reassignment Method. *Transactions on Signal Processing*, 43(5):1068–1089, 1995. ISSN 1053587X. doi: 10.1109/78.382394. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=382394.

Appendix C

Principal Component Analysis

Principal Component Analysis (PCA) maps the input variables – in our case usually a vector of features $\boldsymbol{v}$ – to a new coordinate system by a linear combination of the individual features:

$$\boldsymbol{u}(n) = \boldsymbol{T}^{\mathrm{T}} \cdot \boldsymbol{v}(n). \tag{C.1}$$

The resulting vector $\boldsymbol{u}(n)$ is the data in the new coordinate system for observation n and transformation matrix $\boldsymbol{T}^{\mathrm{T}}$ contains different linear combinations for the input feature vector $\boldsymbol{v}(n)$. The number of features in the vector will be referred to as $\mathcal{F}$. Formulating Eq. (C.1) not only for one observation but also for a series of feature vectors $\boldsymbol{V}$ leads to

$$\boldsymbol{U} = \boldsymbol{T}^{\mathrm{T}} \cdot \boldsymbol{V}. \tag{C.2}$$

The transformation matrix is a square matrix with the dimensions $\mathcal{F} \times \mathcal{F}$. It is composed of vectors defining the linear combinations of the input features:

$$\boldsymbol{T} = \left[\, \boldsymbol{c}_0 \quad \boldsymbol{c}_1 \quad \dots \quad \boldsymbol{c}_{\mathcal{F}-1} \,\right]. \tag{C.3}$$

The transformation matrix has the following main properties:

- the vectors $\boldsymbol{c}_i$ are in the direction of the highest variance in the data and the variance is concentrated in as few output components as possible,
- the vectors $\boldsymbol{c}_i$ are orthogonal to each other

$$\boldsymbol{c}_i^{\mathrm{T}} \cdot \boldsymbol{c}_j = 0 \quad \forall\, i \neq j \tag{C.4}$$

- and the transformation is invertible:

$$\boldsymbol{v}(n) = \boldsymbol{T} \cdot \boldsymbol{u}(n). \tag{C.5}$$

Figure C.1 shows the scatter plots of two hypothetical variables x_1 and x_2 with the original axes on the left and rotated axes on the right.

Figure C.2 shows the application of PCA to the 5 audio features spectral centroid, spectral rolloff, spectral spread, zero crossing rate, peak envelope. The

An Introduction to Audio Content Analysis: Music Information Retrieval Tasks & Applications,
Second Edition. Alexander Lerch.
© 2023 The Institute of Electrical and Electronics Engineers, Inc. Published 2023 by John Wiley & Sons, Inc.

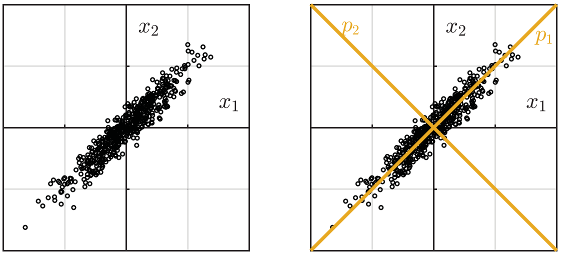
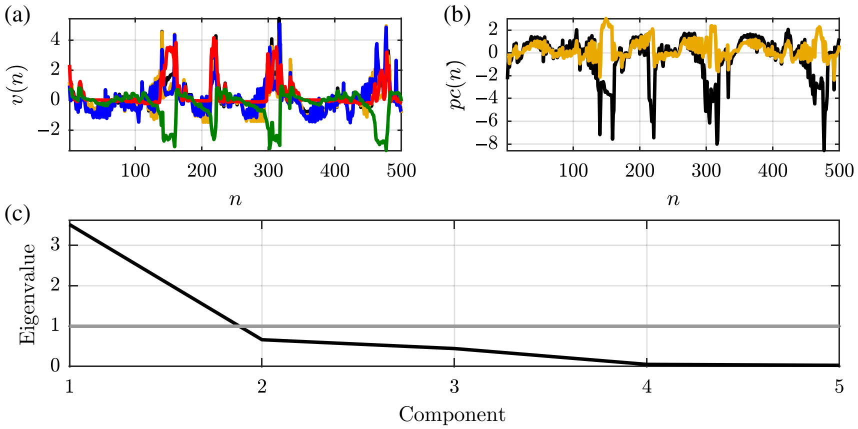


Figure C.1 Scatter plot of a two-dimensional data set with variables x_1, x_2, and the rotated coordinate system after PCA with the component axes p_1, p_2.

Repo 1: plotPca.m



Repo 1: plotPcaExample.m

Figure C.2 Five input features (a) and the two resulting principal components (b); the eigenvalues of the principal components are plotted in (c).

eigenvalues in the bottom clearly show that the two first principal components cover most of the variance in the features.

C.1 Computation of the Transformation Matrix

The first step in computing the matrix $\boldsymbol{T}$ is the calculation of the feature covariance matrix:

$$\boldsymbol{R} = \frac{1}{\mathcal{F}-1} \cdot \left(\boldsymbol{v} - \boldsymbol{\mu}_v\right)\left(\boldsymbol{v}^{\mathrm{T}} - \boldsymbol{\mu}_v^{\mathrm{T}}\right) \qquad \text{(C.6)}$$

with the vector $\boldsymbol{\mu}_v$ containing the arithmetic mean of each feature. The covariance matrix is square, symmetric, and has only positive entries.

The eigenvectors of the covariance matrix represent the axes of the new coordinate system and thus comprise the transformation matrix. Usually, they are ordered with decreasing eigenvalues; the vector in the first column $\boldsymbol{c}_0$ is the vector with the highest eigenvalue and the vector in the last column has the lowest eigenvalue.

C.2 Interpretation of the Transformation Matrix

The transformation matrix $\boldsymbol{T}$ contains useful information on the input data set. Each column is a different linear combination of the input features, and they are sorted according to their eigenvalues or, in other words, according to the variance this component contributes to the overall variance. Features with high influence on the first components can be assumed to be of higher importance than features with high influence on the last components. Thus, PCA can be useful for both feature subset selection and feature space transformation. As the last components have only limited impact on the result, they might be discarded. The matrix $\boldsymbol{T}$ can then be truncated from the dimensions $\mathcal{F} \times \mathcal{F}$ to $\mathcal{F} \times \mathcal{L}$ with $\mathcal{L}$ being the required number of components in the space transformation process.

Appendix D

Linear Regression

Linear regression is a simple approach to regression that assumes a linear relationship between the input features and output. In its simplest form, both the input and the output values are one-dimensional (not vectors). Its application is described in Section 4.2. To arrive at Eqs. (4.2) and (4.3), the goal is to minimize the mean squared error (MSE) (see Eq. (6.7)) between the model and the targets. The minimum can be found by finding where the first derivative of the squared error with respect to both c and m equals 0. For the bias c, we set

$$0 = \frac{\partial}{\partial c}\sum_{r=0}^{\mathcal{R}-1}(y(r) - \hat{y}(r))^2$$

$$0 = \frac{\partial}{\partial c}\sum_{r=0}^{\mathcal{R}-1}(y(r) - mv(r) - c)^2$$

$$0 = \sum_{r=0}^{\mathcal{R}-1} -2\,(y(r) - mv(r) - c)$$

$$c = \frac{\sum\limits_{r=0}^{\mathcal{R}-1} y(r) - m\sum\limits_{r=0}^{\mathcal{R}-1} v(r)}{\mathcal{R}}$$

and finally arrive at

$$c = \mu_y - m \cdot \mu_v.$$

Similarly, the slope m can be determined by finding setting the derivative with respect to m to zero:

$$0 = \frac{\partial}{\partial m}\sum_{r=0}^{\mathcal{R}-1}(y(r) - mv(r) - c)^2$$

$$0 = \sum_{r=0}^{\mathcal{R}-1} - 2\,(y(r) - mv(r) - c) \cdot v(r)$$

An Introduction to Audio Content Analysis: Music Information Retrieval Tasks & Applications,
Second Edition. Alexander Lerch.
© 2023 The Institute of Electrical and Electronics Engineers, Inc. Published 2023 by John Wiley & Sons, Inc.

$$0 = \sum_{r=0}^{\mathcal{R}-1} y(r)v(r) - mv^2(r) - cv(r)$$

$$0 = \sum_{r=0}^{\mathcal{R}-1} y(r)v(r) - mv^2(r) - (\mu_y - m \cdot \mu_v)v(r)$$

$$0 = \sum_{r=0}^{\mathcal{R}-1} y(r)v(r) - mv^2(r) - \mu_y \cdot v(r) + m \cdot \mu_v \cdot v(r)$$

$$0 = \sum_{r=0}^{\mathcal{R}-1} y(r)v(r) - \mu_y \cdot v(r) - m \sum_{r=0}^{\mathcal{R}-1} v^2(r) - \mu_v \cdot v(r)$$

$$m = \frac{\sum_{r=0}^{\mathcal{R}-1} \left(y(r) - \mu_y\right) \cdot v(r)}{\sum_{r=0}^{\mathcal{R}-1} \left(v(r) - \mu_v\right) \cdot v(r)},$$

resulting in

$$m = \frac{\sum_{r=0}^{\mathcal{R}-1} \left(y(r) - \mu_y\right) \cdot \left(v(r) - \mu_v\right)}{\sum_{r=0}^{\mathcal{R}-1} \left(v(r) - \mu_v\right)^2} = \frac{\sigma_{vy}^2}{\sigma_{vv}^2}.$$

Appendix E

Software for Audio Analysis

The common languages and tools can change over time and vary from industry to industry. The desired outcome also influences the language of choice; for example, the requirements are different between prototyping an algorithm and the implementation of "production-quality" software available to customers and users.

In research and prototyping, the language nowadays prevalent is *Python*,[1] often extended by the packages *NumPy*[2] and *SciPy*.[3] The easy extensibility through packages like *scikit-learn*[4] and easy integration of the deep learning packages *TensorFlow*[5] and *PyTorch*[6] make Python often the language of choice especially for machine learning-related tasks. Very common in signal processing is the scripting environment *Matlab*[7] (or the largely compatible open-source software *Octave*[8]) with a simple scripting language and a large set of readily available routines as well as various possibilities of data visualization. Furthermore, there exist visual audio programming environments such as *Max*[9] or *Pure Data (PD)*[10] that have been used in the context of designing real-time audio algorithms.

Many developers consider *C* or *C++* as the language that allows the most workload-efficient implementations. It is not frequently used for algorithmic prototyping as the script-like languages mentioned above allow, due to the inbuilt and available functionality, for more rapid development cycles than a language such as *C++* can provide.

---

[1] Python. https://www.python.org. Last retrieved on 12 December 2021.
[2] NumPy. https://numpy.scipy.org. Last retrieved on 12 December 2021.
[3] SciPy. https://numpy.scipy.org. Last retrieved on 12 December 2021.
[4] scikit-learn. https://scikit-learn.org. Last retrieved on 12 December 2021.
[5] TensorFlow. https://www.tensorflow.org/. Last retrieved on 12 December 2021.
[6] PyTorch. https://pytorch.org. Last retrieved on 12 December 2021.
[7] Matlab. http://www.mathworks.com/products/matlab. Last retrieved on 12 December 2021.
[8] Octave. http://www.octave.org. Last retrieved on 12 December 2021.
[9] Max. http://cycling74.com/products/max. Last retrieved on 12 December 2021.
[10] Pure Data. http://puredata.info. Last retrieved on 12 December 2021.

An Introduction to Audio Content Analysis: Music Information Retrieval Tasks & Applications,
Second Edition. Alexander Lerch.
© 2023 The Institute of Electrical and Electronics Engineers, Inc. Published 2023 by John Wiley & Sons, Inc.

The aim of this appendix is to provide a short overview of software supporting the design and implementation of audio analysis algorithms. The solutions presented below include toolboxes, libraries, frameworks, and applications; however, this list does not claim to be exhaustive and which packages are considered relevant also changes over time. Still, the software presented covers a large variety of Audio Content Analysis (ACA)-related functionality, and each solution focuses on improving or accelerating the prototyping or development process in different areas. Note that we focus on audio-related software only and ignore domain-agnostic frameworks such as scikit-learn, tensorflow, and pytorch.

E.1 Frameworks and Libraries

This section presents software frameworks offering comprehensive possibilities for file input and output, signal processing, audio analysis, and partly even machine learning. Thus, they offer the prototyping and possibly the building of ACA systems.

E.1.1 librosa

At the time of writing, the Python package *librosa*[11] is probably the most commonly used package for audio analysis in Python. Its functionality ranges from wrappers for audio file Input/Output (IO), various preprocessing options, time-frequency transforms, audio feature extraction, and a variety of implementations of various music information retrieval (MIR) tasks [1].

E.1.2 Essentia

Essentia is another widely used library. It is written in *C++* and provides Python bindings.[12] Essentia extracts a wide variety of lower-level and higher-level audio features [2] as well as baseline solutions to a multitude of tasks.

E.1.3 openSMILE

openSMILE is another *C++* framework with Python bindings.[13] It is mainly used for feature extraction but provides several machine learning models for baseline tasks as well. OpenSmile focuses on features used not only in the music domain but also for speech and general audio classification problems [3]. A noteworthy capability of the library is the real-time processing of audio data.

---

[11] librosa. https://github.com/librosa/librosa. Last retrieved on 12 December 2021.

[12] Essentia. https://github.com/MTG/essentia. Last retrieved on 12 December 2021.

[13] openSMILE. https://github.com/audeering/opensmile. Last retrieved on 12 December 2021.

E.1.4 Marsyas

Marsyas is a software framework in *C++* with Python bindings.[14] It offers both real-time and offline processing of audio data as well as audio analysis and synthesis functionality. Since its first publication [4], marsyas has been continuously expanded [5]. The learning curve for new users tends to be steep, but the modular design of the software makes it a versatile tool for many different use cases.

E.1.5 jMIR

The Java framework *j MIR* focuses on the processing of musical instrument digital interface (MIDI) data.[15] The software provides a set of tools for feature extraction and machine learning algorithms as well as tools for metadata annotation [6, 7].

E.1.6 MIRtoolbox

The *MIRtoolbox* is a toolbox for Matlab.[16] It provides a set of functions for the extraction of low-level features as well as for the analysis of tonal, structural, and temporal properties [8].

E.1.7 Yaafe

Yet Another Audio Feature Extractor (YAAFE) is a tool for the extraction of low-level features.[17] It is implemented in *C++* but provides Python bindings as well. YAAFE aims at very efficient feature extraction by utilizing a *feature plan parser*, determining and removing redundant processing steps in the calculation of different features [9].

E.1.8 madmom

madmom is a Python library focusing on onset, beat, and tempo-related tasks.[18] It provides a set of modern solutions to these tasks that are or used to be state-of-the-art (as opposed to baseline implementation featured by many packages mentioned above) [10].

---

[14] Marsyas. https://github.com/marsyas/marsyas. Last retrieved on 12 December 2021.

[15] jMIR. http://jmir.sourceforge.net. Last retrieved on 12 December 2021.

[16] MIRtoolbox. https://www.jyu.fi/hytk/fi/laitokset/mutku/en/research/materials/mirtoolbox. Last retrieved on 12 December 2021.

[17] Yaafe. https://github.com/Yaafe/Yaafe. Last retrieved on 12 December 2021.

[18] madmom. https://github.com/CPJKU/madmom. Last retrieved on 12 December 2021.

E.1.9 Software for Education

Software for education purposes follows different design and implementation principles as software packages that are designed to use out of the box. The software accompanying this book,[19,20] for example, is written neither for efficiency nor for ease of use; rather, the goal was to have consistent, easy to read basic implementations of features and algorithms to enable students and programmers to understand and implement their own adapted versions. Helpful for teaching MIR in the classroom can be the python notebooks released by Müller and Zalkow [11].[21]

E.1.10 Other Software

There are a variety of other audio libraries that can be useful depending on the needs and preferences of the user. A few examples for these are (listed alphabetically) *aubio*,[22] *kapre*,[23] *libXtract*,[24] *Maaate*,[25] *nnaudio*,[26] *pyaudioanalysis*,[27] *spafe*,[28] and *torchaudio*.[29]

E.2 Data Annotation and Visualization

Sonic Visualiser is a software for the visualization of various properties of one or more audio files [12, 13].[30] It is nowadays one of the standard tools for annotating audio recordings, as it offers numerous possibilities to visualize and annotate audio. Compared to the frameworks presented above, it does not focus as much on the algorithm developer but on (nontechnical) expert users such as musicologists who need access to objective information on the audio recording. In order to be able to easily integrate new tools providing analysis data for the visualization, it offers a plugin interface called *VAMP*. The advantage of using plugins for feature extraction is the possibility of dynamically loading new plugins at runtime

---

[19] Matlab: ACA-Code. https://github.com/alexanderlerch/ACA-Code. Last retrieved on 12 December 2021.

[20] Python: pyACA. https://github.com/alexanderlerch/pyACA. Last retrieved on 12 December 2021.

[21] libfmp. https://github.com/meinardmueller/libfmp. Last retrieved on 12 December 2021.

[22] aubio. https://github.com/aubio/aubio. Last retrieved on 12 December 2021.

[23] kapre. https://github.com/keunwoochoi/kapre/. Last retrieved on 12 December 2021.

[24] libXtract. https://github.com/jamiebullock/LibXtract. Last retrieved on 12 December 2021.

[25] Maaate. http://maaate.sourceforge.net/. Last retrieved on 12 December 2021.

[26] nnAudio. https://github.com/KinWaiCheuk/nnAudio. Last retrieved on 12 December 2021.

[27] pyAudioAnalysis. https://github.com/tyiannak/pyAudioAnalysis. Last retrieved on 12 December 2021.

[28] spafe. https://github.com/SuperKogito/spafe. Last retrieved on 12 December 2021.

[29] torchaudio. https://pytorch.org/audio. Last retrieved on 12 December 2021.

[30] Sonic Visualiser. http://www.sonicvisualiser.org. Last retrieved on 12 December 2021.

without compilation or static linking, complemented by the simple exchange of plugins (features) between researchers. A plugin Application Programmer's Interface (API) itself is thus a rather general interface definition of how to extract an unknown feature as opposed to the actual implementation of various features. Other plugin interfaces such as the Feature Extraction Application Programmer's Interface (FEAPI) [14] have not seen widespread use.

References

1 Brian McFee, Colin Raffel, Dawen Liang, Daniel Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and Music Signal Analysis in Python. pages 18–24, Austin, Texas, 2015. doi: 10.25080/Majora-7b98e3ed-003. URL https://conference.scipy.org/proceedings/scipy2015/brian_mcfee.html.

2 Dmitry Bogdanov, Nicolas Wack, Emilia Gómez, Sankalp Gulati, Perfecto Herrera, Oscar Mayor, Gerard Roma, Justin Salamon, José Zapata, and Xavier Serra. ESSENTIA: An Open-source Library for Sound and Music Analysis. In *Proceedings of the ACM International Conference on Multimedia (ACMMM)*, MM '13, pages 855–858, New York, NY, USA, October 2013. Association for Computing Machinery. ISBN 978-1-4503-2404-5. doi: 10.1145/2502081.2502229. URL https://doi.org/10.1145/2502081.2502229.

3 Florian Eyben, Martin Wöllmer, and Björn Schuller. OpenSMILE – The Munich Versatile and Fast Open-Source Audio Feature Extractor. In *Proceedings of the International Conference on Multimedia (MM)*, pages 1459–1462, New York, NY, USA, 2010. ACM Press. ISBN 978-1-60558-933-6. doi: 10.1145/1873951.1874246. URL http://portal.acm.org/citation.cfm?doid=1873951.1874246.

4 George Tzanetakis and Perry Cook. MARSYAS: A Framework for Audio Analysis. *Organised Sound*, 4(3): 169–175, 2000.

5 George Tzanetakis. MARSYAS-0.2: A Case Study in Implementing Music Information Retrieval Systems. In Jialie Shen, John Shepherd, Bin Cui, and Ling Liu, editors, *Intelligent Music Information Systems: Tools and Methodologies*, pages 31–49. IGI Global, 2008.

6 Cory Mckay and Ichiro Fujinaga. jMIR: Tools for Automatic Music Classification. In *Proceedings of the International Computer Music Conference (ICMC)*, Montreal, Quebec, Canada, 2009.

7 Cory Mckay, John Ashley Burgoyne, and Ichiro Fujinaga. jMIR and ACE XML: Tools for Performing and Sharing Research in Automatic Music Classification. In *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries Workshop on Integrating Digital Library Content with Computational Tools and Services*, 2009.

8 Olivier Lartillot and Petri Toiviainen. A Matlab Toolbox for Musical Feature Extraction from Audio. In *Proceedings of the 10th International Conference on Digital Audio Effects (DAFX)*, Bordeaux, 2007.

9 Benoit Mathieu, Slim Essid, Thomas Fillon, Jacques Prado, and Gaël Richard. YAAFE, an Easy to Use and Efficient Audio Feature Extraction Software. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 441–446, 2010.

10 Sebastian Böck, Filip Korzeniowski, Jan Schlüter, Florian Krebs, and Gerhard Widmer. madmom: A New Python Audio and Music Signal Processing Library. In *Proceedings of the ACM International Conference on Multimedia (ACMMM)*, MM ’16, pages 1174–1178, New York, NY, USA, October 2016. Association for Computing Machinery. ISBN 978-1-4503-3603-1. doi: 10.1145/2964284.2973795. URL https://doi.org/10.1145/2964284.2973795.

11 Meinard Müller and Frank Zalkow. libfmp: A Python Package for Fundamentals of Music Processing. *Journal of Open Source Software*, 6(63): 3326, July 2021. ISSN 2475-9066. doi: 10.21105/joss.03326. URL https://joss.theoj.org/papers/10.21105/joss.03326.

12 Chris Cannam, Christian Landone, Mark B Sandler, and Juan Pablo Bello. The Sonic Visualiser: A Visualisation Platform for Semantic Descriptors from Musical Signals. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Victoria, 2006.

13 Chris Cannam, Christian Landone, and Mark Sandler. Sonic Visualiser: An Open Source Application for Viewing, Analysing, and Annotating Music Audio Files. In *Proceedings of the International Conference on Multimedia (MM)*, pages 1467–1468, New York, NY, USA, 2010. ACM Press. ISBN 978-1-60558-933-6. doi: 10.1145/1873951.1874248. URL http://portal.acm.org/citation.cfm?doid=1873951.1874248.

14 Alexander Lerch, Gunnar Eisenberg, and Koen Tanghe. FEAPI: A Low Level Feature Extraction Plugin API. In *Proceedings of 8th International Conference on Digital Audio Effects (DAFX)*, Madrid, September 2005. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2016/10/Lerch-et-al.-2005-FEAPI-A-Low-Level-Feature-Extraction-Plugin-API.pdf.

Appendix F

Datasets

The importance of data in machine learning tasks as well as the importance of availability of reliable ground truth labels have been discussed in Chapter 5. To help the reader, several commonly used datasets are listed in Table F.1. The datasets listed here are selected based on criteria related to availability, popularity, and relevance for the book content; a more complete and up-to-date version of this list is always available at the accompanying website www.AudioContentAnalysis.org.

Table F.1 List of datasets for audio content analysis.

Name	Metadata	Content	Audio
ACPAS [1]	Aligned audio and scores	2 189 performances of 497 scores	✓
ADC2004 [2]	Predominant pitch	20 excerpts	✓
Artist20 [3]	20 artists	1 413 songs	✗
AudioSet [4]	632 event classes	2 084 320 clips (10 s)	✗
AVASPEECH-SMAD [5]	Speech, music	45 h	✓
Bach10 [6]	Multitrack, aligned MIDI	10 chorales	✓
ChoirSet [7]	MIDI, F0, beats	2 songs, 81 takes	✓
Chopin22 [8]	Aligned MIDI	44 recordings	✓
DALI [9]	Aligned notes and lyrics	5 358 songs	✗
DEAM [10]	Valence, arousal	1 802 excerpts	✓
DREANSS [11]	Onset times, perc. instruments	18 excerpts	✓
DrumPt [12]	4 playing techniques	app. 2 000 annotations	✓

(*continued*)

An Introduction to Audio Content Analysis: Music Information Retrieval Tasks & Applications,
Second Edition. Alexander Lerch.
© 2023 The Institute of Electrical and Electronics Engineers, Inc. Published 2023 by John Wiley & Sons, Inc.

Table F.1 (Continued)

Name	Metadata	Content	Audio
EmoMusic [13]	Arousal, valence	744 excerpts (45 s)	✓
EMOPIA [14]	Emotion	1 087 music clips from 387 songs	✓
Emotify [15]	Induced emotion	400 excerpts	✓
ENST-Drums [16]	Onset times, perc. instruments, playing technique	318 segments	✓
FMA [17]	161 genres	106 574 songs	✓
GPT [18]	7 guitar playing techniques	6 580 clips	✓
GTZAN [19]	10 genres	999 excerpts (30 s)	✓
GuitarSet [20]	Midi, pitch, beat, chords	360 guitar excerpts (30 s)	✓
Homburg [21]	9 genres	1 889 excerpts (10 s)	✓
IDMT-SMT-Bass [22]	Bass performance styles	4 300 excerpts	✓
IDMT-SMT-Drums [23]	Onset times, perc. instruments	518 files	✓
IDMT-SMT-Guitar [24]	9 guitar playing techniques	4 700 + 400 note events	✓
IRMAS [25]	11 instruments	2 874 excerpts	✓
JKU-ScoFo [26]	Audio, MIDI	16 recordings	✓
MAESTRO [27]	Audio-aligned MIDI, velocity, sustain, aligned scores	172 h of piano	✓
Magnatagatune [28]	Tags	25 863 excerpts (30 s)	✓
McGill Billboard [29]	Chords	740 songs	✗
MDBDrums [30]	Onset times, perc. instrument, playing technique	23 excerpts	✓
MedleyDB [31]	Multitrack, genre, melody f0, instrument activation	122 songs	✓
MSD [32]	Genre, mood, proprietary features	1 000 000 songs	✗
MTG-Jamendo [33]	Tags (genre, instruments, mood)	55 000 tracks	✓
MTG-QBH [34]	Title, artist	118 queries/481 songs	✓/✗
MUSDB-18 [35]	Multitrack	100 songs	✓

(*continued*)

Table F.1 (Continued)

Name	Metadata	Content	Audio
MusicNet [36]	Pitch and onsets	330 recordings	✓
NSynth [37]	Instrument and pitch	305 979 single notes	✓
OpenBMAT [38]	6 classes for music presence	1 647 excerpts (60 s)	✓
OpenMIC-2018 [39]	20 instruments	20 000 excerpts (10 s)	✓
RWC [40]	Lyrics, genres, chords, structure, aligned MIDI	115 songs/50 classical/100 songs	✓
SMD [41]	Audio-aligned MIDI	50 recordings	✓
SoundTracks [42]	Valence, energy, tension, mood	360 + 110 excerpts	✓
TONAS [43]	Pitch	72 single-voiced excerpts	✓
UrbanSound8k [44]	10 event classes	8 732 slices	✓
VocalSet [45]	17 vocal techniques	3 560 recordings	✓
WJazzD [46]	Onset, pitches	456 Jazz solos	✗

References

1 Lele Liu, Veronica Morfi, and Emmanouil Benetos. ACPAS: A Dataset of Aligned Classical Piano Audio and Scores for Audio-To-Score Transcription. In *Late Breaking Demo (Extended Abstract), Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, volume 162, page 113769, Online, 2021. doi: 10.1016/j.eswa.2020.113769. URL https://linkinghub.elsevier.com/retrieve/pii/S0957417420305935.

2 Graham E Poliner and Daniel P W Ellis. A Classification Approach to Melody Transcription. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 161–166, London, 2005. doi: 10.7916/D81Z4DT7. URL https://doi.org/10.7916/D81Z4DT7.

3 Daniel P W Ellis. Classifying Music Audio with Timbral and Chroma Features. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Vienna, Austria, September 2007. doi: 10.5281/zenodo.1416906. URL https://zenodo.org/record/1416906.

4 Jort F Gemmeke, Daniel P W Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R Channing Moore, Manoj Plakal, and Marvin Ritter. Audio Set: An Ontology and Human-labeled Dataset for Audio Events. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*

(ICASSP), pages 776–780, New Orleans, LA, March 2017. IEEE. ISBN 978-1-5090-4117-6. doi: 10.1109/ICASSP.2017.7952261. URL http://ieeexplore.ieee.org/document/7952261/.

5 Yun-Ning Hung, Karn N Watcharasupat, Chih-Wei Wu, Iroro Orife, Kelian Li, Pavan Seshadri, and Junyoung Lee. AVASPEECH-SMAD: A Strongly Labelled Speech and Music Activity Detection Dataset with Label Co-occurrence. In *Late Breaking Demo (Extended Abstract), Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 3, Online, 2021.

6 Zhiyao Duan, Bryan Pardo, and Changshui Zhang. Multiple Fundamental Frequency Estimation by Modeling Spectral Peaks and Non-Peak Regions. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(8):2121–2133, November 2010. ISSN 1558-7924. doi: 10.1109/TASL.2010.2042119. Conference Name: IEEE Transactions on Audio, Speech, and Language Processing.

7 Sebastian Rosenzweig, Helena Cuesta, Christof Weiß, Frank Scherbaum, Emilia Gómez, and Meinard Müller. Dagstuhl ChoirSet: A Multitrack Dataset for MIR Research on Choral Singing. *Transactions of the International Society for Music Information Retrieval (TISMIR)*, 3(1):98–110, 2020. doi: 10.5334/tismir.48. URL http://doi.org/10.5334/tismir.48. Publisher: Ubiquity Press.

8 Werner Goebl. Melody Lead in Piano Performance: Expressive Device or Artifact? *The Journal of the Acoustical Society of America*, 110(1):563–572, July 2001. ISSN 0001-4966. doi: 10.1121/1.1376133. URL https://asa.scitation.org/doi/abs/10.1121/1.1376133.

9 Gabriel Meseguer-Brocal, Alice Cohen-Hadria, and Geoffroy Peeters. DALI: A Large Dataset of Synchronized Audio, Lyrics and notes, Automatically Created using Teacher-student Machine Learning Paradigm. In Emilia Gómez, Xiao Hu, Eric Humphrey, and Emmanouil Benetos, editors, *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 431–437, Paris, France, 2018. URL http://ismir2018.ircam.fr/doc/pdfs/35_Paper.pdf.

10 Anna Aljanaki, Yi-Hsuan Yang, and Mohammad Soleymani. Developing a Benchmark for Emotional Analysis of Music. *PLoS ONE*, 12(3):e0173392, March 2017. ISSN 1932-6203. doi: 10.1371/journal.pone.0173392. URL https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0173392. Publisher: Public Library of Science.

11 Ricard Marxer and Jordi Janer. Study of Regularizations and Constraints in NMF-based Drums Monaural Separation. In *Proceedings of the International Conference on Digital Audio Effects (DAFX)*, page 6, Maynooth, 2013.

12 Chih-Wei Wu and Alexander Lerch. On Drum Playing Technique Detection in Polyphonic Mixtures. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, New York, 2016. ISMIR. doi: 10.5281/zenodo.1416680. URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2016/07/Wu_Lerch_2016_On-Drum-Playing-Technique-Detection-in-Polyphonic-Mixtures.pdf.

13 Mohammad Soleymani, Micheal N Caro, Erik M Schmidt, Cheng-Ya Sha, and Yi-Hsuan Yang. 1000 Songs for Emotional Analysis of Music. In *Proceedings of the 2nd ACM International Workshop on Crowdsourcing for Multimedia - CrowdMM '13*, pages 1–6, Barcelona, Spain, 2013. ACM Press. ISBN 978-1-4503-2396-3. doi: 10.1145/2506364.2506365. URL http://dl.acm.org/citation.cfm?doid=2506364.2506365.

14 Hsiao-Tzu Hung, Joann Ching, Seungheon Doh, Nabin Kim, Juhan Nam, and Yi-Hsuan Yang. EMOPIA: A Multi-Modal Pop Piano Dataset For Emotion Recognition and Emotion-based Music Generation. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Online, August 2021. URL http://arxiv.org/abs/2108.01374.

15 Anna Aljanaki, Frans Wiering, and Remco C Veltkamp. Studying Emotion Induced by Music through a Crowdsourcing Game. *Information Processing & Management*, 52(1):115–128, January 2016. ISSN 0306-4573. doi: 10.1016/j.ipm.2015.03.004. URL https://www.sciencedirect.com/science/article/pii/S0306457315000424.

16 Olivier Gillet and Gaël Richard. ENST-Drums: An Extensive Audio-Visual Database for Drum Signals Processing. In *ISMIR*, Victoria, 2006. URL http://ismir2006.ismir.net/PAPERS/ISMIR0627_Paper.pdf.

17 Michaël Defferrard, Kirell Benzi, Pierre Vandergheynst, and Xavier Bresson. FMA: A Dataset For Music Analysis. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Suzhou, China, September 2017. URL http://arxiv.org/abs/1612.01840.

18 Li Su, Li-Fan Yu, and Yi-Hsuan Yang. Sparse Cepstral and Phase Codes for Guitar Playing Technique Classification. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 6, Taipei, Taiwan, 2014.

19 George Tzanetakis, Georg Essl, and Perry Cook. Automatic Musical Genre Classification Of Audio Signals. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Bloomington, 2001.

20 Qingyang Xi, Rachel M Bittner, Johan Pauwels, Xuzhou Ye, and Juan P Bello. GUITARSET: A Dataset for Guitar Transcription. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 8, Paris, France, 2018.

21 Helge Homburg and Ingo Mierswa. A Benchmark Dataset for Audio Classification and Clustering. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 4, London, UK, 2005.

22 Jakob Abeßer, Hanna Lukashevich, and Gerald Schuller. Feature-based Extraction of Plucking and Expression Styles of the Electric Bass Guitar. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2290–2293, Dallas, March 2010. doi: 10.1109/ICASSP.2010.5495945. ISSN: 2379-190X.

23 Christian Dittmar and Daniel Gärtner. Real-time Transcription and Separation of Drum Recordings based on NMF Decomposition. In *Proceedings of the International Conference on Digital Audio Effects (DAFX)*, page 8, Erlangen, Germany, 2014.

24 Christian Kehling, Jakob Abeer, Christian Dittmar, and Gerald Schuller. Automatic Tablature Transcription of Electric Guitar Recordings by Estimation of Score- and Instrument-related Parameters. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, page 8, Erlangen, Germany, 2014.

25 Juan J Bosch, Jordi Janer, Ferdinand Fuhrmann, and Perfecto Herrera. A Comparison of Sound Segregation Techniques for Predominant Instrument Recognition in Musical Audio Signals. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 6, Porto, Portugal, 2012.

26 Florian Henkel, Stefan Balke, Matthias Dorfer, and Gerhard Widmer. Score Following as a Multi-Modal Reinforcement Learning Problem. *Transactions of the International Society for Music Information Retrieval*, 2(1):67–81, November 2019. ISSN 2514-3298. doi: 10.5334/tismir.31. URL http://transactions.ismir.net/articles/10.5334/tismir.31/.

27 Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset. In *Proceedings of the International Conference on Learning Representations (ICLR)*, New Orleans, LA, January 2019. URL http://arxiv.org/abs/1810.12247.

28 Edith Law, Kris West, and Michael Mandel. Evaluation of Algorithms Using Games: The Case of Music Tagging. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 6, Kobe, Japan, 2009.

29 John Ashley Burgoyne, Jonathan Wild, and Ichiro Fujinaga. An Expert Ground-Truth Set for Audio Chord Recognition and Music Analysis. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 6, Miami, FL, 2011.

30 Carl Southall, Chih-Wei Wu, Alexander Lerch, and Jason A Hockman. MDB Drums — An Annotated Subset of MedleyDB for Automatic Drum

Transcription. In *Late Breaking Demo (Extended Abstract), Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Suzhou, 2017. International Society for Music Information Retrieval (ISMIR). URL http://www.musicinformatics.gatech.edu/wp-content_nondefault/uploads/2017/10/Wu-et-al_2017_MDB-Drums-An-Annotated-Subset-of-MedleyDB-for-Automatic-Drum-Transcription.pdf.

31 Rachel Bittner, Justin Salamon, Mike Tierney, Matthias Mauch, Chris Cannam, and Juan Bello. MedleyDB: A Multitrack Dataset for Annotation-Intensive MIR Research. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 7, Taipei, Taiwan, 2014.

32 Thierry Bertin-Mahieux, Daniel P W Ellis, Brian Whitman, and Paul Lamere. The Million Song Dataset. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Miami, FL, October 2011. Columbia University. doi: 10.7916/D8NZ8J07. URL https://www.ee.columbia.edu/dpwe/pubs/BertEWL11-msd.pdf.

33 Dmitry Bogdanov, Minz Won, Philip Tovstogan, Alastair Porter, and Xavier Serra. The MTG-Jamendo Dataset for Automatic Music Tagging. In *Machine Learning for Music Discovery Workshop, International Conference on Machine Learning (ICML 2019)*, page 3, Long Beach, 2019.

34 Justin Salamon, Joan Serrà, and Emilia Gómez. Tonal Representations for Music Retrieval: From Version Identification to Query-by-Humming. *International Journal of Multimedia Information Retrieval*, 2(1):45–58, March 2013. ISSN 2192-662X. doi: 10.1007/s13735-012-0026-0. URL https://doi.org/10.1007/s13735-012-0026-0.

35 Zafar Rafii, Antoine Liutkus, Fabian-Robert Stöter, Stylianos Ioannis Mimilakis, and Rachel Bittner. MUSDB18 - A Corpus for Music Separation. Technical report, 2017. URL https://zenodo.org/record/1117372.

36 John Thickstun, Zaid Harchaoui, and Sham M Kakade. Learning Features of Music From Scratch. In *Proceedings of the International Conference on Learning Representations (ICLR)*, page 14, Toulon, France, 2017.

37 Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Mohammad Norouzi, Douglas Eck, and Karen Simonyan. Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders. In *Proceedings of the International Conference on Machine Learning*, pages 1068–1077. PMLR, July 2017. URL https://proceedings.mlr.press/v70/engel17a.html. ISSN: 2640-3498.

38 Blai Meléndez-Catalán, Emilio Molina, and Emilia Gómez. Open Broadcast Media Audio from TV: A Dataset of TV Broadcast Audio with Relative Music Loudness Annotations. *Transactions of the International Society for Music Information Retrieval*, 2(1):43–51, August 2019. ISSN 2514-3298. doi: 10.5334/tismir.29. URL http://transactions.ismir.net/articles/10.5334/tismir.29/. Number: 1 Publisher: Ubiquity Press.

39 Eric J Humphrey, Simon Durand, and Brian McFee. OpenMIC-2018: An Open Dataset for Multiple Instrument Recognition. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 7, Paris, France, 2018.

40 Masataka Goto. RWC Music Database: Popular, Classical, and Jazz Music Databases. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, page 2, Paris, 2002.

41 Meinard Müller, Verena Konz, Wolfgang Bogler, and Vlora Arifi-Müller. Saarland Music Data (SMD). In *Late Breaking Demo (Extended Abstract), Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Miami, FL, 2011. Place: Miami, USA.

42 Tuomas Eerola and Jonna K Vuoskoski. A Comparison of the Discrete and Dimensional Models of Emotion in Music. *Psychology of Music*, 39(1):18–49, August 2010. ISSN 0305-7356. doi: 10.1177/0305735610362821. URL http://pom.sagepub.com/cgi/doi/10.1177/0305735610362821.

43 Emilia Gómez and Jordi Bonada. Towards Computer-Assisted Flamenco Transcription: An Experimental Comparison of Automatic Transcription Algorithms as Applied to A Cappella Singing. *Computer Music Journal*, 37(2):73–90, June 2013. ISSN 0148-9267. URL https://doi.org/10.1162/COMJ_a_00180.

44 Justin Salamon, Christopher Jacoby, and Juan Pablo Bello. A Dataset and Taxonomy for Urban Sound Research. In *Proceedings of the ACM International Conference on Multimedia (ACMMM)*, MM ’14, pages 1041–1044, New York, NY, USA, November 2014. Association for Computing Machinery. ISBN 978-1-4503-3063-3. doi: 10.1145/2647868.2655045. URL https://doi.org/10.1145/2647868.2655045.

45 Julia Wilkins, Prem Seetharaman, Alison Wahl, and Bryan Pardo. VocalSet: A Singing Voice Dataset. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pages 468–474, Paris, France, 2018. URL http://ismir2018.ircam.fr/doc/pdfs/114_Paper.pdf.

46 Martin Pfleiderer, Klaus Frieler, Jakob Abeßer, Wolf-Georg Zaddach, and Benjamin Burkhart, editors. *Inside the Jazzomat - New Perspectives for Jazz Research*. Schott Campus, 2017.

Index

a

A Weighting 222
ACA 1, 2, 4, 11, 12, 14, 17, 37, 71, 72, 97, 107, 112, 163, 168, 170, 175, 186, 194, 225, 231, 245, 253, 255, 260, 281, 319–321, 325, 337, 355, 367, 412
Accuracy 119, 120, 122, 123, 165, 186, 242, 249, 326, 341, 342, 349
ACF 3, 62, 63, 145–148, 152, 153, 156, 159, 198, 244, 245, 247, 251, 380–382
ACF Maximum 62, 63
Acoustic Onset Time, *see* AOT
Activation Function 261
ADT 260–262
Aeolian Mode 170
Aeolic Mode 170
Aliasing 29, 369, 392
Alignment Path 282–284, 288–290, 293, 294
Alternative Blackman Window 397
AMDF 147–149
AMDF-Weighted Autocorrelation Function 148
Amplitude 400
Angular Frequency 18
ANN 324, 348
AOT 230, 231
API 415
Area Under Curve, *see* AUC
Arithmetic Mean 20, 22, 23, 25–27, 58–60, 64, 68, 168, 242, 244, 245, 406
Articulation 236, 340
Artificial Neural Network, *see* ANN
Attack 230
Attack Time 42, 223, 230, 348
aubio 414
AUC 123, 124
Audio Content 11
Audio Content Analysis, *see* ACA
Audio Fingerprinting 312
Audio Signal 17
  Periodic 18
  Random 20
Audio-to-Audio Alignment 281, 289–291, 293, 295
Audio-to-Score Alignment 281, 291, 293, 294
Auditory Filterbank 37, 38
Autocorrelation Function, *see* ACF
Automatic Drum Transcription, *see* ADT
Average Magnitude Difference Function, *see* AMDF

An Introduction to Audio Content Analysis: Music Information Retrieval Tasks & Applications, Second Edition. Alexander Lerch.
© 2023 The Institute of Electrical and Electronics Engineers, Inc. Published 2023 by John Wiley & Sons, Inc.

b

Balanced Accuracy 123
Bar 179, 235, 251, 252
Bar Length 251, 252
Bark 44, 129, 130, 225
Bark Scale 131
Bartlett Window 396
Baseline 120
Beat 179, 186, 232, 233, 235, 245–251, 255, 381
Beat Detection 250, 251
Beat Grid 245–247, 249
Beat Histogram 39, 243–245, 247, 322
Beat Period 246
Beat Phase 245, 247, 248, 250
Beat Spectrum 243, 244
Beat Strength 244
Beat Tracking 247–250
Beats per Minute, *see* BPM
Blackman Window 396
Blackman-Harris Window 396
Block Length 29–31, 33, 41, 140, 144, 220, 221, 225, 237, 309, 379, 399
Block Overlap Ratio 30, 31, 225, 239, 309
Block-Based Processing 29
Box-Cox Transform 71
BPM 232, 233, 235, 245
BS.1770 222, 225
BS.468 222

c

C Weighting 222
C++ Framework for Audio and Music, *see* CLAM 222
CASA 3
CCF 181, 185, 247, 248, 251, 252, 378–382
CCIR Weighting 222
CD Quality 14
Cent 136
Center Clipping 147
Center of Gravity, *see* COG
Central Moment 20, 46
Centroid 23, 182, 348
Cepstrum 52, 146, 152–154
Chord 133, 187–191, 195
Chord Detection 195
Chord Inversion 187, 195
Chord Progression 191
Chord Recognition 60, 168, 179, 187, 188, 190, 191, 194, 195, 246
Chord Template 188, 189
Chord Type 195
Chorus 252
Chroma Perception 132
Chromatic Mode 170
CiCF 181, 380, 397
Circle of Fifths 172, 173, 181, 183, 185, 191–193
Circular Correlation Function, *see* CiCF
Classification 52, 73, 74, 319, 320, 322, 323, 341
Clustering 99
CNN 68, 97, 237, 324, 348
Coefficient of Determination 125, 341
COG 43, 239
Comb Filter 158
Comité Consultatif International des Radiocommunications, *see* CCIR
Compact Disc, *see* CD
Composition 11, 355
Computational Auditory Scene Analysis, *see* CASA
Computer Audition 3
Concert Pitch 166
Confusion Matrix 122
Constant *Q* Transform, *see* CQT
Content-based Audio and Music Extraction Library, *see* CAMEL
Convolution 152, 372–375, 381, 386–387, 392, 395, 397
  Associativity 373
  Circularity 374, 375
  Commutativity 373
  Distributivity 374

Identity 372
Convolutional Neural Network, *see* CNN
Correlation 74, 189, 191, 233, 308
Correlation Coefficient
Correlation Function 62, 145, 148, 152, 378–380
Cosine Distance 101, 185, 238, 244, 290, 319
Cosine Similarity 101, 181
Cosine Window 396, 398
Cost Matrix 284–288
Covariance Matrix 406
CQT 35, 37, 169, 179
Critical Band 131, 237
Critical Band Rate 44, 129, 131, 132
Cross Correlation Function, *see* CCF
Cross Validation 110, 111
Cut-Off Frequency 158, 376, 377

d
Data 107, 108, 119
Data Augmentation 111, 112, 192
Data Split 119
Davies-Bouldin-Index 126
DC Removal 27
DCT 52
Decibel 54, 217, 218, 371
Decision Tree 95
Deep Neural Network, *see* DNN
Delta Function 372, 390–392, 395
Delta Pulse 158, 247, 248, 252, 392
Descriptor, *see* Feature
DFT 29, 31–33, 35, 52, 375, 385, 386, 397–399, 402
Diatonic Scale 133
Diatonic Temperament 137
Differentiator 27, 28
Digital Signal Processing, *see* DSP
Dimensionality Reduction 72, 73, 76
Discrete Cosine Transform, *see* DCT
Discrete Fourier Transform, *see* DFT
Disk Jockey, *see* DJ
Distance Matrix 282–290
Distance Measure 101
  Cosine 101
  Euclidean 101
  Manhattan 101
DJ 246, 253, 317
DNN 36, 97, 99, 109, 163, 182, 186, 237, 249
Dominant 187
Dominant Chord 187
Dorian Mode 170
Down-Mixing 26, 27, 143, 307
Down-Sampling 29, 289, 307
Downbeat 235, 249–252
Downbeat Detection 251, 253
DP 248, 284, 292–294
DSP 2
DTW 193, 194, 248, 281, 282, 286–290, 293, 294, 296
Dynamic Programming, *see* DP
Dynamic Time Warping, *see* DTW
Dynamics 12, 219, 291, 356

e
EM 94
Embeddings 14
Emotion 337–339
Enharmonic Equivalence 133, 135, 136, 172, 185
Envelope 40, 223, 224, 237, 244, 290, 307, 381
Envelope Extraction 12
Epoch 320
Equal Temperament 135–137, 169, 180
Equal-Loudness Contour 221, 222
Equivalent Rectangular Bandwidth, *see* ERB
ERB 132
Essentia 412
Euclidean Distance 56, 70, 93, 101, 160, 185, 290, 308, 319, 342
European Broadcasting Union, *see* EBU
Evaluation 119

Expectation Maximization, *see* EM
Expected Value 22

f

F-beta Score 123
F-Measure 123, 165, 242, 249, 250, 259, 326, 341, 349, 350
F-score, *see* F-Measure
F1 score, *see* F-Measure
False Negative, *see* FN
False Negative Rate, *see* FNR
False Positive, *see* FP
False Positive Rate, *see* FPR
Fast Fourier Transform, *see* FFT
FEAPI 415
Feature 3, 13, 14, 17, 31, 37, 39–42, 52, 62, 63, 67–75, 92, 97, 100, 107, 181, 191, 217, 220, 237, 245, 251, 255, 256, 290–294, 307, 308, 318, 319, 322–324, 340, 348, 349, 359, 405, 406, 412–415
Feature Aggregation 68
Feature Extraction Application Programmer's Interface, *see* FEAPI
Feature Learning 66, 114, 115
Feature Matrix 67, 68
Feature Representation 14
Feature Scaling 70
Feature Space 100, 101, 319
Feature Space Transformation 73, 75, 76, 407
Feature Subset Selection 73, 407
FFT 33, 35, 382, 399
Filter 375
Filterbank 37
Fingerprint 305–308, 310
Fingerprinting 305–307, 309
Finite Impulse Response, *see* FIR
FIR 158, 375–377, 387
Floating Point 371
FN 122, 123, 241–243, 307
FNR 123, 312
Fourier Series 18, 138, 385, 397
Fourier Transform, *see* FT
FP 122–124, 239, 241–243, 259, 307
FPR 123, 124, 242, 312
Frequency Reassignment 142
FT 32, 34, 52, 157, 377, 382, 385–386, 388–392, 395–398
  Convolution 386
  Frequency Scaling 390
  Frequency Shift 388
  Multiplication 386
  Parseval's Theorem 387
  Superposition 386
  Symmetry 388
  Time Scaling 390
  Time Shift 388
Full Scale 218, 220, 224, 371
Full-Wave Rectification, *see* FWR
Fundamental Frequency 3, 18, 20, 43, 47, 65, 129, 138–140, 142–144, 149–152, 154–159, 163, 180, 292, 293, 322, 348, 385
Fundamental Frequency Detection 65, 138, 139, 143, 157, 247, 399
FWR 244

g

Gammatone Filter 38
Gammatone Filterbank 38, 155
Gaussian Distribution 21
Gaussian Mixture Model, *see* GMM
Gaussianity 47, 71
Geometric Mean 22, 23, 58
Gibbs' Phenomenon 20
GMM 94–96, 120, 192, 318, 324, 348
Ground Truth 107, 122, 164–166, 170, 194, 241, 258, 259, 294, 295

h

Half-Wave Rectification, *see* HWR
Hamming Window 396

Harmonic Mean 23
Harmonic Mixing 174
Harmonic Mode 170
Harmonic Product Spectrum, *see* HPS
Harmonic Sum Spectrum, *see* HSS
Harmonics 18, 36, 42, 47, 51, 58, 129, 138, 139, 143, 144, 149–152, 154, 155, 158, 180, 348
Harmony 129, 143, 187, 251, 322
HFC 45
Hidden Markov Model, *see* HMM
High Frequency Content, *see* HFC
High-Pass Filter 67
Hit Rate 312
HMM 96, 190, 191, 292–294
HMM Toolkit, *see* HTK
Homophony 143
Hop Size 29–31, 220, 221, 309, 401, 402
HPS 149–151, 159, 196
HSS 151, 158, 196
HTK 54, 55
HWR 57, 155, 158, 159, 237, 238, 290

i

IBI 233, 248, 249
ICA 76
IDFT 397
IFT 386, 389, 391
IIR 68, 226, 375, 377
Impulse Response 38, 158, 372, 373, 375, 377, 386, 387
Independent Component Analysis, *see* ICA
Inference 91
Infinite Impulse Response, *see* IIR
Initial Transient Time 230
Input Representation 14, 237
Input/Output, *see* IO
Instantaneous Feature 26, 39, 40, 174, 220, 322
Instantaneous Frequency 62, 142, 169, 202, 238, 399–402
Instrumentation 320
Integration Time 220, 223
Intensity 217, 219, 220, 322
Inter-Beat Interval, *see* IBI
Inter-Onset Interval, *see* IOI
International Organization for Standardization, *see* ISO
International Society for Music Information Retrieval, *see* ISMIR
International Telecommunication Union, *see* ITU
Interval 134, 136, 182, 187
Intonation 12, 137, 138
Inverse Discrete Fourier Transform, *see* IDFT
Inverse Fourier Transform, *see* IFT
IO 412
IOI 229, 231–233, 247, 251
Iterative Subtraction 157

j

jMIR 413
JNDL 218
Just Noticeable Difference in Level, *see* JNDL

k

K-Means Clustering 99, 318
K-Nearest Neighbor, *see* K-NN
K-NN 93, 94, 101, 120, 324
kapre 414
Key 57, 129, 133, 137, 169–174, 181–185, 187, 189, 191, 318, 340
Key Detection 12, 60, 168, 170, 179, 182, 186, 187
Key Profile 182–185, 191
Key Signature 172
Kullback-Leibler Divergence 102, 160
Kurtosis 46, 47, 71, 245

l

L1 Distance 101
L2 Distance 101
Laplace Distribution 21
Latency 30, 289, 380
LDA 76, 290
Leave-One-Out Cross Validation 74, 111, 324
Leptokurtic 47
librosa 412
libXtract 414
Linear Discriminant Analysis, *see* LDA
Linear Regression 98, 409
Local Tempo 233, 235
Log-Mel Spectrogram 36, 37, 39, 97
Lokrian Mode 170
Loss Function 160
Loudness 28, 42, 44, 175, 217–221, 225, 294, 318, 322, 338
Loudness Range 225, 322
Low-Level Feature 40, 64, 318, 413
Low-Pass Filter 67, 375
Lydian Mode 170
Lyrics 320

m

MA 27, 60, 68, 78, 220, 239, 240, 375, 376
Maaate 414
Machine Learning, *see* ML
Machine Listening 3
Macro Accuracy 123
madmom 413
MAE 124, 125, 165, 242, 341
Magnitude Spectrogram 176
Magnitude Spectrum 33, 43–47, 49, 51, 52, 57–62, 142, 149, 152–154, 179, 243, 245, 309, 382, 385, 388, 389
Main Lobe 63, 395, 398
Major Mode 133, 170, 172, 183–185
Manhattan Distance 56, 101, 308, 310
Marsyas 413
Mean Absolute Error, *see* MAE
Mean Squared Error, *see* MSE
Mean Tempo 233
Meantone Temperament 137
Median 22, 24, 25, 240
Median Filter 240
Mel 52, 54, 130
Mel Frequency Cepstral Coefficient, *see* MFCC
Mel Scale 36, 37, 52, 55, 129–132
Mel Spectrogram 36, 37, 255, 323, 324
Mesokurtic 47
Metadata 1, 91, 305, 306, 317, 338, 413
Meter 229, 232–235, 248, 250–252
Metrics 119
MFCC 51, 52, 54, 55, 71, 120, 255, 290, 291, 307, 318, 322, 324, 340
Micro Accuracy 123
Mid-Level Feature 318
MIDI 2, 111, 135, 136, 164, 191, 219, 230, 247, 291–295, 413
Min-Max Normalization 70
Min-Max Scaling, *see* Min-Max Norm.
Minor Mode 170, 183–185
MIR xvii, 2, 13, 121, 325, 412, 414
MIRtoolbox 413
Mixolydian Mode 170
ML 91, 107, 121
Mode 170, 172, 184
Modulation 171, 173, 182
Monophonic 143, 144
Mood 337–341
Mood Recognition 97, 337, 338, 340, 341
Motion Picture Experts Group, *see* MPEG
Moving Average, *see* MA
MP3 310
MPA 357, 358
MPEG-1 Layer 3, *see* MP3

MSE 98, 104, 121, 124, 125, 165, 242, 409
Multi-Label 348, 349
Multi-Pitch Detection 157–159, 180, 181
Multitask Learning 114
Music Emotion Recognition 337
Music Genre 13, 317
Music Genre Classification 321, 325
Music Informatics 3
Music Information Retrieval, *see* MIR
Music Information Retrieval Evaluation eXchange, *see* MIREX
Music Performance 234, 290–292, 305, 338, 355–358
Music Performance Analysis, *see* MPA
Music Performance Assessment 355, 358, 359
Music Recommendation 317
Music Similarity 317–319
Musical Communication 357
Musical Dynamics 219
Musical Form 234, 320
Musical Genre 317, 319–322, 324, 326
Musical Genre Classification 68, 319–322, 324, 337, 340
Musical Instrument Digital Interface, *see* MIDI
Musical Instrument Recognition 347–349
Musical Key 185, 186
Musical Score 359

n

Nearest Neighbor 74, 93
NMF 159–163, 196, 262
nnAudio 414
Noisiness 59, 65
Non-negative Matrix Factorization, *see* NMF
Normalization 28, 54, 56, 70, 71, 220, 379
NOT 230, 231
Note 235, 251, 355
Note Onset Time, *see* NOT
Note Value 235, 236
Novelty Function 236–241, 243–248, 251, 252, 256, 261

o

Observation 24, 67, 70
Octave 133
Offset Time 235
Onset 229–231, 236–239, 241, 242, 245, 248, 251, 260, 261, 293, 294
Onset Detection 57, 92, 231, 232, 236–243, 256, 260–262, 347
Onset Time 229–232, 235–237, 239, 241–243, 248, 290, 292, 293, 295
openSMILE 412
Outlier 24
Overfitting 72, 97, 102, 103, 114
Overshoot 19, 20
Overtone 138

p

P-score 249, 250
PAT 231
PCA 75, 76, 81, 405
PDF 20–22, 24, 25, 46, 47, 292, 370, 371
Peak Envelope 98, 405
Peak Picking 236, 239, 240, 247, 261, 262
Peak Program Meter, *see* PPM
Peak Structure Distance, *see* PSD
Pearson Correlation Coefficient 101, 380
Perceived Tempo 233
Perceptual Attack Time, *see* PAT
Perceptual Onset Time, *see* POT
Period 252

Period Length 18, 139, 140, 143–145, 154, 156, 158, 234, 385, 400
Periodicity 20
Phase 142, 400
Phase Spectrum 385, 388, 389, 399, 400
Phrase 252
Phrygian Mode 170
Pitch 3, 42, 129, 132–138, 155, 164, 168, 248, 307, 355
Pitch Chroma 71, 174–177, 179–187, 189, 191, 192, 194, 251, 255, 290, 293
Pitch Chromagram 176, 177
Pitch Class 132, 133, 136, 170, 172, 174–176, 181
Pitch Detection 138, 195
Pitch Height 130, 132, 166
Pitch Histogram 323
Pitch Tracking 138, 164, 168
Platykurtic 47
Playlist Generation 317
Polyphonic 143, 144, 156, 322
POT 231
Power Spectrum 44, 46, 49, 55, 59, 146, 385
PPM 223, 224, 226
Pre-Whitening 146, 159
Precision 92, 123, 124, 240, 242, 326, 341
Predominant Melody Extraction 157
Preprocessing 26
Principal Component Analysis, *see* PCA
Probability Density Function, *see* PDF
Probe Tone Ratings 183–185
Process Loss 398, 399
Production 12
PSD 292, 293
pyAudioAnalysis 414
Pythagorean Temperament 137

q

Quantile 24
Quantile Range 24, 25
Quantization 367, 369, 371
Quantization Error 370, 371
Quantization Step Size 370
Quartile 25

r

Random Forest 95, 96, 120
Raw Chroma Accuracy 165
Raw Pitch Accuracy 165
Real-Time 28, 30, 293, 413
Recall 92, 123, 124, 240, 242, 326, 341
Receiver Operating Characteristic Curve, *see* ROC
Rectangular Window 391, 395–396
Rectilinear Distance 101
Recurrent Neural Network, *see* RNN
Regression 97, 99
Relative Frequency Distribution, *see* RFD
Release Time 223, 224
Representation Learning 66
Rest 235, 236
Revised Low-Frequency B Curve, *see* RLB
RFD 22–24
Rhythm 229, 233, 234, 239, 318, 320, 340
RLB Weighting 222, 225
RMS 24, 28, 93, 98, 199, 219–222, 224, 225, 227, 237, 255, 290, 291, 322, 324, 325, 340
RNN 237
ROC 124, 242
Root Mean Square, *see* RMS
Root Note 133, 187, 195

s

Sample Rate 29, 33, 63, 139, 140, 143, 239, 307, 309, 367–369, 392
Sample Rate Conversion 29, 143
Sampling 367
Sampling Theorem 29, 369, 392
Scale Degree 132, 137, 138, 170, 187

Score 2, 11, 12, 133, 134, 138, 169, 172, 234–236, 291, 292, 294, 338, 355, 356, 358
Score Following 291–293
Selectivity 123
Self Similarity Matrix, *see* SSM
Self-Organizing Map, *see* SOM
Sensitivity 123
Sentence 252
Sequential Backward Elimination 74, 75
Sequential Forward Selection 74, 75
Sharpness 44
Short Time Fourier Transform, *see* STFT
Side Lobe 395, 398
Signal-to-Noise Ratio, *see* SNR
Silhouette Coefficient 125
SIMD 30
Similarity Matrix 244, 251, 286
Singing Voice Detection 347
Single Instruction Multiple Data, *see* SIMD
Single-Pole Filter 221, 376
Singular Value Decomposition, *see* SVD
Skewness 46, 47, 71, 245
Smoothing 68, 239, 376
Smoothing Filter 375
SNR 371
Software Developer’s Kit, *see* SDK
SOM 319
Sonic Visualiser 414
spafe 414
Sparse Coding 66
Specificity 123
Spectral Centroid 26, 43–48, 98, 307, 322, 324, 325, 340, 405
Spectral Crest Factor 57, 58, 307
Spectral Decrease 50
Spectral Envelope 50, 51, 146, 159, 318
Spectral Flatness 58–60, 307
Spectral Flux 55, 56, 238, 290
Spectral Kurtosis 47, 48
Spectral Leakage 395, 399
Spectral Rolloff 48, 49, 405
Spectral Skewness 47
Spectral Slope 50, 51
Spectral Spread 45–48, 405
Spectrogram 34, 72, 157, 160, 163, 262
Spectrum 29, 52, 54, 59, 60, 150, 152–154, 157, 158, 174, 385, 388, 390–393, 395, 397
SSM 251, 253–256, 258, 265, 282
Standard Deviation 23, 24, 57, 68, 70, 165, 168, 182, 231, 232, 242, 245
Standardization 70
STFT 33–37, 48, 55, 60, 62, 140–142, 150, 152, 153, 158, 169, 174, 175, 179, 237–239, 244, 290, 309, 310, 386, 393, 397, 400, 401
Stratified Sampling 109
Structure 253
Structure Detection 246, 251, 253, 259
Subdominant Chord 187
Subfeature 68, 323
Support Vector Machine, *see* SVM
Support Vector Regression, *see* SVR
SVD 76
SVM 95, 99, 120, 324, 340, 348
SVR 99, 340

t

Tactus 232, 243, 252
Tatum 233, 234, 251, 252
Temperament 136, 137, 166, 169
Tempo 11, 26, 229, 232, 233, 235, 239, 245–251, 288, 290, 292, 293, 295, 307, 318, 322, 338, 340
  Main 233
  Mean 233
  Mode 233
Tempo Detection 12, 236, 246, 247, 249, 250
Test Data 70, 109, 110, 119
Texture Window 31, 68, 69, 182, 220, 244, 323

Timbre 12, 42, 43, 46, 54, 144, 166, 175, 219, 294, 318–320, 322
Time Signature 233, 235, 251, 252, 322
Time Stamp 31
Timing 11, 231, 234, 288, 290, 291
TN 122, 242
TNR 123
Tonal Centroid 185
Tonal Power Ratio 60
Tonality 57, 185
Tonalness 57, 59, 60, 62, 179
Tonic 137, 170, 172, 183
Tonic Chord 187, 191
torchaudio 414
TP 122, 242, 259, 307
TPR 123, 124, 242, 312
Training Data 70, 109–111
Transfer Learning 66, 113, 114
Transient 19, 45, 59, 220, 230
Tremolo 12, 219, 243
Triad 187–189
True Negative, *see* TN
True Negative Rate, *see* TNR
True Peak Meter 225
True Positive, *see* TP
True Positive Rate, *see* TPR
Tuning Frequency 136, 166–170, 177
Tuning Frequency Estimation 166, 168, 170
Type I error, *see* FP
Type II error, *see* FN

u

Underfitting 102

v

Validation Data 109
Variance 23, 24, 46, 76, 156, 295, 405, 407
Velocity 219
Verse 252
Vibrato 12, 138, 219, 243
Viterbi Algorithm 97, 188–193, 248, 281
Vocal Activity 165
Vocal Activity Detection 347
von-Hann Window 396, 397

w

Watermarking 305, 306
Waveform 34
Wholetone Mode 170
Wiener-Khinchin Theorem 381
Window
  Alternative Blackman 397
  Bartlett 396
  Blackman 396
  Blackman–Harris 396
  Cosine 396
  Hamming 396
  Rectangular 396
  von-Hann 396, 397
Window Function 375, 391, 393, 395, 396, 398
Word Length 308, 370, 371, 373
Workload 30, 32, 287, 311, 382, 411

y

YAAFE 413
Yet Another Audio Feature Extractor, *see* YAAFE 413

z

Z-Score Normalization 70, 324
Zero Crossing Rate 64, 65, 144, 292, 322, 405
Zero Phase Filtering 377
Zero-Padding 142, 397
Zero-R-Classifier 120

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley’s ebook EULA.